FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Route Optimization of Airships Taking into Account the Meteorological Properties of the Atmosphere

Niklas Heidenreich

Masterarbeit

Route Optimization of Airships Taking into Account the Meteorological Properties of the Atmosphere

Niklas Heidenreich

Masterarbeit

Aufgabensteller:	Prof. Dr. Christoph Pflaum
Betreuer:	Prof. Dr. Christoph Pflaum
Bearbeitungszeitraum:	01.8.2024 – 03.02.2025

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Masterarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 29. Januar 2025 .....

Abstract

In recent years, airships have seen a resurgence in interest as a more sustainable alternative to airplanes or ships for the transportation of passengers and goods. They offer a more efficient and environmentally friendly way of transportation as they have the potential to be powered by solar energy alone. Creating a carbon-neutral way of transportation that is also able to move large loads over long distances. Companies are developing new airship technologies that are able to carry more cargo over greater distances with the goal of creating a zero-carbon footprint airship.

With transportation being the only sector that is still increasing its carbon footprint, the need for such technologies is more pressing than ever. This thesis will focus on simulating the theoretical power output generated by an airship that uses solar panels as its primary energy source. Such an airship would be capable of transporting large loads while maintaining carbon neutrality. The new model will employ an advanced raytracer to enhance the precision of solar radiation calculations and will utilize real-world weather data from the MERRA-2 dataset to assess solar radiation under various weather conditions and altitudes [1].

Contents

1 Introduction **7**

2 Goal **8**

3 Overview of Airships **9**
3.1 Advantages of airships . . . 9
3.2 Solar powered airships . . . 9

4 Solar area calculation **10**
4.1 File format . . . 10
4.2 Basic mesh implementation . . . 10
4.3 Calculation of the solar area using ray tracing . . . 11
4.3.1 Creating the image plane . . . 11
4.3.2 Calculating Triangle hits . . . 12
4.3.3 Rendering the mesh . . . 12
4.3.4 Adding the solar placement angle . . . 14
4.3.5 Problems with the solar placement angle . . . 14
4.4 Fast Calculation of the solar area . . . 15
4.5 Optimization: Culling of triangles . . . 15
4.5.1 Cull direction facing triangles . . . 16
4.5.2 Cull occluded triangles . . . 16
4.6 Comparison to the old model . . . 16
4.7 Caching the Solar area . . . 17
4.8 Result comparison . . . 17

5 Solar radiation **19**
5.1 Calculation of the solar radiation . . . 19
5.2 Reference model . . . 19
5.3 LibRadtran . . . 20
5.4 Comparison of model results . . . 20
5.5 Integrating LibRadtran into airship pathfinder software . . . 21
5.5.1 Calling the Radtran simulation from C++ . . . 22
5.5.2 Performance using the LibRadtran model . . . 22
5.5.3 Caching the radiation values . . . 23
5.5.4 Parallelizing the cache building process . . . 25

6 Simulation of solar radiation with weather data **26**
6.1 Combining the weather data with the radiation simulation . . . 26
6.1.1 Weather data (Merra-2) . . . 26
6.1.2 LibRadtran file format for weather data . . . 27
6.1.3 Extracting and reading the data from the Merra-2 dataset . . . 28
6.1.4 Using the weather data in the airship simulation . . . 29
6.2 Calculating radiation using weather data . . . 30
6.2.1 Radtran with weather data . . . 30
6.2.2 Diffuse radiation . . . 30
6.2.3 Solar area calculation with diffuse radiation . . . 31
6.2.4 Diffuse radiation on the underside of the airship . . . 31

7 Resulting calculations of the routefinder **33**
7.1 Example flight from London to New York . . . 33
7.2 Cloud dependent flight altitude . . . 34
7.3 Theoretical maximal radiation during the flight . . . 35
7.4 Upward difussive radiation during the flight . . . 36
7.5 Circumventing clouds . . . 37
7.6 Performance problems for low irradiance routes . . . 38

8 Solar panel efficiency per wavelength **40**

9 Diffuse radiation **43**
9.1 Irradiance vs radiance . . . 43
9.2 Calculation of the diffuse radiation . . . 43
9.2.1 Interpolation . . . 45
9.2.2 Lebedev Integration . . . 46
9.2.3 Precision of the Lebedev quadrature . . . 48
9.2.4 Directional diffusive irradiance per frequency . . . 48

10 Approximating the LibRadtran results using a neural network **50**
10.1 Generating the data . . . 50
10.2 Predicting the irradiances . . . 50
10.3 Predicting the radiances . . . 52

11 Summary **54**
11.1 Conclusion . . . 54
11.2 Outlook . . . 55

References **56**

Acronyms **57**

1 Introduction

Reducing the current carbon footprint to mitigate the effects of climate change is one of the most pressing issues of our time. Significant research is being done to develop new technologies that are both sustainable and cost-effective in nearly every sector of the economy. From more sustainable generation of electricity to ever more efficient production of goods, the demand for new innovative technologies is everywhere. However, while all sectors of the economy are being optimized, the transportation sector is still not only lagging behind but is also the only sector that is still increasing its carbon footprint. From 1990 to 2019, the emissions produced by the transportation sector increased by 33% [2]. This is only going to get worse as the global population increases, the tourism sector grows, and the demand for global transportation of goods becomes ever more important.

This massive transportation industry is not only responsible for producing a large amount of CO2 but is also a significant contributor to noise pollution. According to the EEA report [2], an estimated 18.4 million people are exposed to transport-related noise levels that are harmful to their health.

Airships represent a promising returning technology that could not only help to reduce the carbon footprint of the transportation sector but also allow for decreased noise pollution. They are able to carry large loads over long distances and can do so with a fraction of the fuel consumption of a plane. The biggest downside of airships is that they are very slow compared to planes, which may limit their use for the long-distance transportation of passengers but not for the transportation of goods. The most famous airship, the Hindenburg, made a journey from Lakehurst, New Jersey, to Frankfurt, Germany, in just over 43 hours. This speed is generally sufficient for transporting goods and is still considerably faster than shipping by sea. Average shipping duration times from Germany to the US by express air freight are around 3 days [3]. The actual flight time is around 10 hours, and the rest of the time is used for loading and unloading the plane, customs, and other bureaucratic procedures. As a result, the flight time represents only a tiny fraction of the time it takes to transport goods. This gets even more extreme when considering the average non-express air freight shipping time, which is around 6 days. In these cases, it would be hardly noticeable if the flight time doubled or even tripled.

Currently, companies like LTA Research are developing new airship technologies that are able to carry more cargo over greater distances. Their airship, the "Pathfinder 1", features a rigid body structure instead of an inflatable one. Such a structure would allow solar panels to be installed on the airship, which could be used to power the whole aircraft and achieve a zero carbon footprint.

Airships are a prime candidate for solar-powered flight as they feature massive surface areas for solar panels and are efficient enough to be powered by solar energy alone. And while the added weight of solar panels would be a problem for planes, it is less of an issue for airships. As, increasing the airship's size to accommodate more solar panels would also significantly enhance its cargo capacity. The main issue with solar energy is its strong reliance on weather conditions.

This thesis will focus on implementing a sophisticated solar radiation model. It will utilize high-resolution weather data to calculate the solar radiation that reaches the airship, as well as the solar power that can be generated from it.

2 Goal

The goal of this thesis is to improve an existing airship route finding software by implementing a more sophisticated model for the solar radiation reaching the airship during its flight. Not only will the new model be able to approximate the solar radiation more accurately than the old model, but it will also be able to use the provided weather data to calculate solar radiation in different realistic weather conditions. This improvement is achieved through the use of an advanced ray tracer combined with real-world weather data. The obtained radiation data can then be used to calculate the power that can be generated by solar panels on the airship during a given flight. Using this data allows for more informed flight route decisions and may help the airship to find more optimal routes. The solar panel output may be better in certain weather conditions or different altitudes at certain times of the day. The airship pathfinding algorithm can then use this information to find a more optimal route.
The featured model is also capable of simulating the irradiance on the airship on a per-wavelength basis. By considering the efficiency of the solar panels at different wavelengths, the model is able to calculate the power that can be generated by them more accurately. This can be especially useful as most solar panels are not pointing directly at the sun but are angled and, therefore, may experience a shift in the wavelength of the incoming radiation. All this data combined further allows for a more extensive view into different possible solar panel configurations and their absorption spectra.

3 Overview of Airships

3.1 Advantages of airships

Air transport based on lighter-than-air technology allows for a near energy-free flight ability. Such vehicles are able to remain airborne for extended periods of time without the need for fuel, as their ability to stay aloft is based on the principle of buoyancy rather than the principle of lift. Much like a ship floating on water, an airship floats in the air and, in principle, needs no energy to maintain its altitude. The only energy that is needed is for movement and steering. With proper route planning, even this movement energy can be minimized by using the wind to move the airship. The near energy-free flight ability combined with the usage of wind for movement is one of the greatest strengths of airships. However, this reliance on wind currents also presents a significant challenge that hinders their competitiveness with airplanes. Not only does this extreme dependence on the wind make the airship very slow, but it also makes route planning highly unpredictable. If the wind is in the wrong direction or too strong, the airship may not be able to move at all, or flight becomes impossible entirely.
However, even with these problems sounding detrimental to the use of airships, one should not forget that the first successful transatlantic flight using an airship was made in 1919 by the airship R34 of the British Royal Air Force. This remarkable feat demonstrates that airships have been able to overcome these obstacles already more than 100 years ago. During this transatlantic flight, the airship encountered winds of over $80km/h$ and still managed to reach its destination [4].
There are also other advantages of airships that make them a very interesting alternative to airplanes. For example, they are significantly quieter than airplanes and, therefore, produce less noise pollution. This is not only a benefit for the people inside the airship but also for the people living on the ground. The noise pollution of airplanes is a significant problem for people living near airports, and airships could help reduce this problem.
While they may not be as efficient as trains for the transportation of people, they do not require any ground infrastructure such as rails or roads to operate. This makes them significantly more flexible and cheaper than trains, especially in regions where the infrastructure is not as developed or hard to develop. They could also be used to resupply remote regions that are hard to reach by other means of transportation or assist in disaster relief efforts.
Additionally, airships present a compelling option for medium-range passenger transportation. While they may not be as fast as airplanes, they are significantly more comfortable and are able to provide an excellent view of the landscape. People are willing to pay significantly more for first-class seats on an airplane than for economy-class seats, indicating a strong preference for comfort during travel. Most airships feature a large windowed cabin that allows for a great view of the landscape, therefore not only giving a more comfortable flight experience but also a more interesting one.
For shorter intracontinental flights, where the time difference between an airplane and an airship is not as significant, airships could, therefore, be a very interesting alternative to airplanes. Because of their calmer flight experience, airships could help people who are afraid of flying or elderly people who may find it challenging to travel by plane.

3.2 Solar powered airships

Airships have a very large surface area, which has historically remained largely unused. However, with modern advancements in solar panel technology, this surface area may be utilized to install solar panels that generate the energy needed to power the airship. This would allow for a carbon-neutral way of transporting goods or people. Because of the large surface area of the airship, it is possible to install enough solar panels to power the whole airship during the entire trip. Battery technology has also improved significantly in recent years, and it is now possible to store enough energy to power the airship throughout the night.
During the day, the airship could regenerate these batteries using the solar panels. A similar approach is already being developed by LTA Research for their Pathfinder airship series. Their airship design also features extra fuel cells that can be used to generate energy during the night or when the solar panels are not generating enough energy. Currently, nearly all research on solar-powered airships is related to rigid-body airships. Rigid-body airships feature a rigid structure in their hull that can hold its shape without the need for gas pressure. In contrast, non-rigid-body airships, also called blimps, need gas pressure for their hull to achieve its shape. While flexible solar panels could be used on blimps, rigid-body airships can use more efficient and durable solar panels.

4 Solar area calculation

Airships require minimal energy to stay aloft as they fly using the principle of buoyancy. The only energy that is needed is for movement and steering. A byproduct of this reliance on buoyancy is that airships need a gigantic volume of gas, mostly helium or hydrogen, to make the whole craft lighter than air. Such a massive volume features a massive surface area that is, in its current state, not used for anything other than keeping the gas inside the airship. The low energy consumption of airships, combined with the large surface area that is not used for anything else, makes airships a prime candidate for solar-powered flight. As this surface area is primarily the hull of the airship it can not be seen as simply a big flat surface; it consists of multiple differently angled surfaces of varying sizes. In order for the solar panels to generate power, they need to be exposed to the sun. It is, therefore, important for the accuracy of the simulation to calculate the correct solar area for any given sun position. Solar panels that are not illuminated by the sun, such as those facing away from it or those occluded by other parts of the airship, will not contribute to power generation. It is further necessary to not only get the solar area for all possible sun positions, but for all directions in general. Diffusive radiation also plays a role in the power generation of the solar panels, as it can reach the airship from various angles, including from the ground. The algorithm designed for this calculation should be both accurate and fast, as it needs to be executed numerous times during the airship's pathfinding simulation.

4.1 File format

The 3D model of the airship is stored in the .stl file format. The file format was already used in the previous version of the Pathfinder software, so there is no need to change it. Using this file format allows for easy parsing using C++. It is also lightweight, as using this file format requires no additional libraries. The data is stored directly as triangles along with their corresponding normal vector, which provides all the necessary information for computing the airship's solar area. Using the binary version of the .stl file format can significantly reduce the file size without sacrificing ease of parsing.

```
Header 80 bytes
Number of triangles 4 bytes
foreach triangle
  Normal vector 12 bytes
  Vertex 1 12 bytes
  Vertex 2 12 bytes
  Vertex 3 12 bytes
  Attribute byte count 2 bytes
```

The airship model utilized in this thesis is based on the LZ-129 Hindenburg [5]. This model was selected due to the airship being well known and it consisting of a rigid-body. Additionally, it is one of the largest airships ever constructed, making it an excellent candidate for calculations related to solar-powered flight. There is also a wealth of data available on this airship, including its dimensions and weight.

4.2 Basic mesh implementation

To enable the existing software to easily access the new 3D data of the airship, a new Mesh class has been implemented. The Mesh class has to read the .stl file and store the mesh's vertices and faces in a way that is easy to access. As neither the order of the vertices nor their color is important, the .stl file will simply be read triangle for triangle and store them in a std::vector. Each triangle is defined as a struct with three vertices and a normal vector, both of which are also extracted from the .stl file.

```
struct Vec3 {
  float x,y,z;
};
struct Triangle3 {
  Vec3 a,b,c;
  Vec3 normal;
};
class Mesh{
  std::vector<Triangle3> triangles;
public:
```

```
    ...
}
```

4.3 Calculation of the solar area using ray tracing

A highly accurate method for calculating the solar area is to use ray tracing. This technique, however, is very computationally expensive, and therefore, a lightweight version of a ray tracer is used. This simpler method does not utilize more advanced techniques like anti-aliasing or depth of field. In this approach, instead of generating multiple rays for each pixel—where each ray would start at a defocus disc and end at various points around the target pixel—only one ray is generated for each pixel. This ray begins at the center of the pixel and travels in the direction of the sun's rays until it hits the airship. As we only care if the ray hits the airship, we do not need to check for the orientation of the triangle that the ray hits. It could point away from the sun and still be hit by the ray, but as we expect the airship to be a closed mesh, this is not a problem. Implementation-wise, this is done by iterating over every pixel in the image and then checking if the ray hits the airship. To check that, we iterate over every triangle in the mesh and check if the ray, starting at the given pixel coordinate, intersects with it. If it does, we increment the hit counter by one. In the end, one can divide the hit counter by the total number of rays to get the percentage of the image where the airship is hit by the rays. If one now knows the area that the image represents, one can calculate the area of the airship that is hit by the rays.

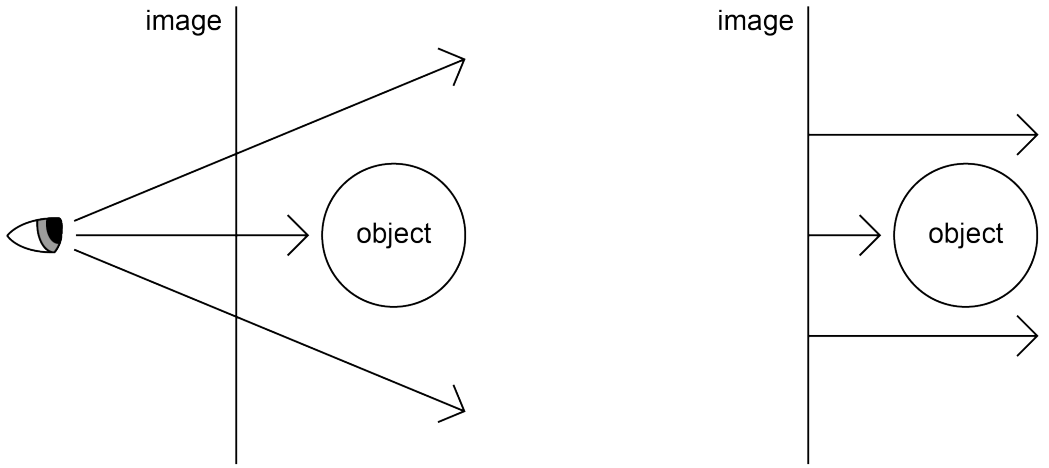

Figure 1: Left a typical way to implement the camera in a ray tracing algorithm. Right the way the camera is implemented in this algorithm. This implementation leads to interesting side effects like the size of the airship in the image no longer being dependent on the distance to the camera.

Knowing the area of the image that is used for raytracing the airship is significantly easier when the raytracers camera is implemented with a few deviations from the default camera implementations. This means that instead of having a point of origin and a direction vector to a point on a plane in front of it that represents the resulting image, one now simply has a 2D plane of specified size and traces rays from every pixel on that plane in the direction of the planes normal vector. An illustration of this concept can be found in Figure 1. This camera implementation ensures that the size of the airship in the image is not influenced by the distance to the camera, but rather solely by the dimensions of the image plane. By now knowing the size of the image plane in meters, one can continue to calculate the area of the airship that is hit by the rays.

4.3.1 Creating the image plane

As the raytracing algorithm only returns the percentage of the image that is hit by the rays, it is necessary to know the size of the image plane in meters in order to calculate the area of the airship that is hit by the

rays. The raytracing function gets the ray origin and the ray direction as input. It then tries to create a plane that is perpendicular to the ray direction, passes through the ray origin, and is big enough to cover the whole airship. Also, because of how the camera is implemented, the size of the plane is not dependent on the distance to the camera, nor is the size of the airship in the image. [Figure 1] Therefore, the distance between the plane and the airship can be chosen arbitrarily as long as the plane is far enough away from the airship to avoid any intersections. A simple approach to creating such a plane is as follows:

```
void calculate_plane(Vec3 plane_center_origin, Vec3 plane_normal, float max_length,
↪  int x_res, int y_res) {
  // Try to find the two vectors that are perpendicular to the ray-direction that
↪  create the plane
  double a,b,c;
  a = plane_normal.y;
  b = -plane_normal.x;
  c = 0;

  // If the plane_normal is pointing in the z-direction, we have to choose a
↪  different plane
  if (length(Vec3(a,b,c)) < 1e-2) {
      a = plane_normal.z;
      b = 0;
      c = -plane_normal.x;
  }

  Vec3 plane_direction_1 = normalize(Vec3(a,b,c));
  Vec3 plane_direction_2 = normalize(cross(plane_normal, plane_direction_1));

  // Calculate the top left corner of the plane and the step size in x and y
↪  direction
  Vec3 top_left = plane_center_origin - plane_direction_1 * max_length / 2 -
↪  plane_direction_2 * max_length / 2;
  Vec3 step_x = plane_direction_1 * max_length / x_res;
  Vec3 step_y = plane_direction_2 * max_length / y_res;
  // ...
}
```

This method is not perfect, but it is sufficient for the purpose of this algorithm. It will create a plane that is perpendicular to the plane_normal and goes through the planer_center_origin. It also uses the length parameter to determine the top left corner of the plane and the step size in the x and y directions.

4.3.2 Calculating Triangle hits

In order for the raytracer to work it is necessary to implement a method that checks if a given ray will hit a triangle in the 3D mesh. For this, the method will simply iterate over every triangle in the mesh and then check if the ray intersects with it. This can be improved by using more sophisticated algorithms, like quadtrees or bounding boxes, to locate promising hit targets more easily, but for the purpose of this quick area calculation, these are not necessary. Instead, in order to calculate if a ray hits a triangle, simply the Möller-Trumbore intersection algorithm is used. This algorithm was selected due to its simplicity and speed. It does not require any special preprocessing steps for different values, making it easy to maintain, update, or extend. Algorithms using precomputed values may be faster but are also more complicated. With its current use case, real-time raytracing of high-poly models is simply not needed, so the speed of the algorithm is not a critical factor, and the added complexity is not justified.

4.3.3 Rendering the mesh

In order to see a visual representation of the airship, one can render the result of the ray tracing algorithm. This is done by iterating over every pixel in the image and checking if the ray hits the airship. If a ray hits the airship, the pixel is colored black; if it misses, the pixel is colored white. So, instead of adding all hits together and returning a single value, the algorithm can instead also just return the calculated image. This results in a black-and-white image where the airship is represented by black pixels.
If one, instead of coloring the pixels that represent airship hits black, colors them by the angle between the normal of the triangle that the ray hits and the direction of the sun rays, one can see the angle at which

the sun rays hit the airship. This results in an image that shows the airship in a more shaded way, with the parts that are hit more directly being brighter than the parts that are hit at an angle. An example can be seen in Figure 3. The resulting array is then written in .ppm format to a file. This file format was chosen as it is straightforward to implement without the usage of any additional libraries. It also preserves the original values from the array since .ppm files do not use compression algorithms.

```
  // render shaded image
  value = 255;
  if(hit){
    mesh::Vec3 normal = image[i].value().normal();
    double angle = acos(mesh::dot(normal, camera_direction) / (normal.length() *
↪   camera_direction.length())) * 180 / std::numbers::pi;
    value = angle/360.0 * 255;
  }
```

Figure 2: The code that is used to render the shaded image.

Figure 3: Left a representation of the internal hit/ no-hit state of the raytracer used to calculate the solar are. Right a shaded render of the reference airship model. Rendered by the code shown in Figure 2.

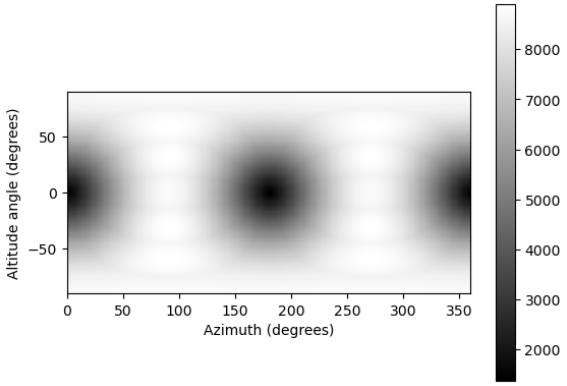

Figure 4: The calculated solar area $[m^2]$ for the airship with given altitude and azimuth angles as viewing directions.

4.3.4 Adding the solar placement angle

In reality, not every part of the airship is equipped with solar panels. Therefore, it is necessary to calculate the solar area that is actually utilized by the solar panels instead of just calculating the whole area of the airship that is hit by the rays. Currently, the airship pathfinding program uses a simple way of describing the position of the solar panels on the airship called the "solar placement angle." This angle indicates how the solar panels are oriented on the airship. For instance, a solar placement angle of $90\,\mathrm{deg}$ would mean that all triangles that have a normal vector with a component pointing in the upward direction will have solar panels on them. This can be seen in Figure 5.
Therefore, any triangle with an angle less than a certain threshold is considered to be part of the solar area. This approximation allows for a quick selection of areas that are most likely to face the sun, making it an acceptable first estimation for potential solar panel placement. It is also possible to run the computation multiple times with different thresholds to simulate different kinds of solar panels at different angles.

Figure 5: Shaded render of the reference airship model with all parts black that are not relevant to the radiation calculation. I.e. they do not have solar panels installed. Here the "solar placement angle" is set to $90\,\mathrm{deg}$. This means that all triangles of the mesh that have a normal vector with a component pointing in the upward direction will have solar panels.

4.3.5 Problems with the solar placement angle

Currently, the calculation of the solar placement angle will check every visible triangle to see if its orientation angle is within the bounds defined by the placement angle. However, this approach has a significant drawback: some sections of the airship may contain very few triangles, which is particularly true for the hull.
Instead of the curvature of the airship being a continuous circle, it is divided into a handful of sections, each with a certain degree of rotation. Once the solar placement angle hits a specified point, it suddenly adds or subtracts a sizable portion of the airship's solar area, causing an abrupt change without any gradual transition in-between. This can be seen in Figure 6. Fortunately, this issue resolves itself because rigid-body airships have internal structures that mimic this angled behavior. Therefore, any interpolation technique aimed at smoothing the differences between varying surface angles would likely reduce the accuracy of the simulation instead.

Figure 6: Left the solar cells at a solar placement angle of 44 and right at 45 degrees.

4.4 Fast Calculation of the solar area

While the raytracing algorithm is very accurate, it is also very slow. A major contributing factor to this slow speed is the unnecessarily high runtime cost that arises when increasing the image's precision. This is because the algorithm has to check every triangle in the mesh for every pixel in the image. As a result, the runtime complexity is $O(n \cdot m_x \cdot m_y)$ where n is the number of triangles in the mesh and m_x, m_y are the numbers of pixels in the image in the x and y directions, respectively.
Instead of a pixel-based approach that artificially limits the accuracy of the solution by the resolution of the image, one can instead use a triangle-based approach. This faster algorithm to calculate the solar area reverses the idea of the raytracing version. Instead of checking hits on triangles for every pixel in an image, one can iterate over every triangle in the list and add its area times the cosine of the angle between the inverted viewing direction and the triangle normal. So the resulting area $A_{\text{direction}}$ can be calculated as follows:

$$A_{\text{direction}} = A_{\text{full}} \cdot \cos \alpha$$

with A_{full} being the area of the triangle and α being the angle between the inverted viewing direction and the triangle's normal vector. Which can be calculated using the following formula:

$$\cos \alpha = \frac{\langle n, -d \rangle}{\|n\| \|d\|}$$

Here, n is the triangle's normal vector, and d is the viewing direction vector defined by the azimuth and altitude angles. This algorithm is significantly faster than the raytracing algorithm as it only requires iterating over each triangle in the mesh once. Additionally, the area calculations for the triangles are also very efficient and can even be performed in advance. As a result, the algorithm primarily focuses on calculating the cosine of the angle between the viewing direction and the triangle's normal vector. This algorithm's runtime complexity is $O(n)$, with n being the number of triangles in the mesh. A significant improvement over the raytracing algorithm.

4.5 Optimization: Culling of triangles

Both the raytracing algorithm and the triangle-based algorithm depend heavily on the number of triangles in the mesh for their runtime. One effective optimization technique applicable to both algorithms is to cull the triangle data in advance, retaining only the triangles that are necessary for the computation. Reducing the number of triangles results in fewer collision checks, leading to improved performance. Ideally, this process leaves only the triangles required for calculating the solar area.

4.5.1 Cull direction facing triangles

The first and most simple technique for cutting triangles is to remove all triangles that are not facing the correct direction. If the inverted normal vector of a triangle has no component facing in the same direction as the viewing direction, then we will just see the backside of this triangle, which can safely be ignored. Doing this reduces the number of triangles to be checked for the sidewards view of our airship from 304 to 152. A significant reduction in triangles for nearly no overhead as the normal vector check is very fast, and the runtime complexity of the algorithm is $O(n)$.

```
std::vector<Triangle3> Mesh::get_triangles_direction(const Vec3& ray_direction) const
↪  {
  std::vector<Triangle3> triangles_direction;
  for (const Triangle3& t : _triangles) {
      if (dot(t.normal(), ray_direction) < 0) {
          triangles_direction.push_back(t);
      }
  }
  return triangles_direction;
}
```

4.5.2 Cull occluded triangles

I also implemented another culling technique that can be used to cull the number of triangles even further. All occluded triangles can safely be ignored. Unfortunately, finding these triangles is a bit more resource intensive than a simple normal vector check. Triangles do not have to be fully occluded but can be partially occluded by multiple other triangles. One approach, called "get_triangles_visible_center()", will trace a ray from the center of every triangle in the inverse direction of the viewing direction and check for hits on other triangles. This is not only significantly slower than merely checking the triangles' facing direction but is also too aggressive in its culling. Some triangles—particularly larger ones—can be culled simply because their centers are obstructed, even if part of them is still visible. A culling technique that over-culls is, therefore, worse than one that under-culls and misses a few irrelevant triangles. The other version called "get_triangles_visible_corners()" only culls if all corners are obstructed. This means that there are still some edge cases where all corners are occluded, but some parts of the triangle are not. Though these are exceedingly rare and have such little visible area that they do not amount to much. An airship mesh seen from the side uses 304 triangles unculled, 152 culled by only looking at the direction, 144 if checked for center occlusion, and 152 if checked for corner occlusion. Ultimately, both occlusion-checking techniques require significantly more resources to cull than the simple direction technique while providing no real benefits. As this version needs to check every triangle for a hit on every other triangle, the runtime complexity is $O(n^2)$.

4.6 Comparison to the old model

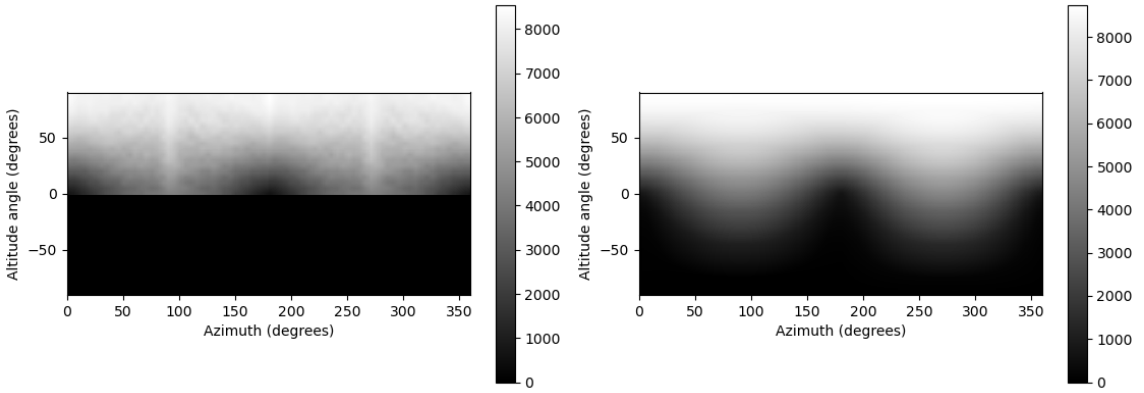

Figure 7: Solar area at a solar placement angle of 90 degrees. Left the old version. Right the new version.

In the airship pathfinder program there already existed an implementation of a solar area algorithm using the solar placement angle [5]. After experimenting with it, however, it was concluded that something was wrong with the output of this already-existing version. As can be seen in Figure 7, the new implementation appears significantly more realistic. It exhibits fewer artifacts and is overall smoother than the previous version. Additionally, the old version was unable to calculate the solar area for altitude angles less than zero, a limitation that has now been addressed in the new implementation.

4.7 Caching the Solar area

The solar area is checked millions of times during the runtime of a single simulation. Recomputing it each time consumes a significant amount of computing power on something that is only dependent on a simple direction vector. To speed up the process of obtaining the correct solar area data, a cache has been implemented that will calculate a subset of all possible solar areas and interpolate between them. The calculation of the solar area is solely dependent on the view direction. This direction can be broken down into an azimuth angle and a solar zenith angle (sza). These two values will then be iterated over in $1\,\mathrm{deg}$ steps, and the solar area will be calculated. As the range for the angles is defined (azimuth angle is $[0, 360]$ and for sza is $[-90, 90]$), we know how many entries the cache will have. This setup means that accessing the cache is extremely fast since no complex calculations are necessary—only a simple array lookup is required. Values not in the cache can be interpolated using neighboring array values. Building the cache only takes roughly a second, after which all following solar area requests are extremely fast.

4.8 Result comparison

Implementation	Calculated Area [m^2]
Raytracing (64x64)	9340.96
Raytracing (4096x4096)	8711.58
Triangle based	8722
Triangle based + Direction culling	8722
Triangle based + Center Culling	8712
Triangle based + Corner Culling	8722

Table 1: The calculated solar areas of the different algorithms for the side view of the airship.

Figure 8: The airship model as seen from the side. Left for a resolution of 64×64 pixels and right for 2048×2048 pixels. The left image results in a measured solar area of $9340.96m^2$ while the right image results in a measured solar area of $8723.9m^2$.

The raytracing algorithm is highly dependent on the resolution of the generated image. As a result, it has high calculation times as the complexity of calculating an $x \times y$ image of an airship model consisting of n triangles is roughly $O(xyn)$. This means that the algorithm is not only slow but also inaccurate except for very high resolutions.
In contrast, the triangle-based algorithm is significantly faster and offers comparable accuracy. Therefore, in the final version of the Pathfinder software, the triangle-based approach is utilized.

5 Solar radiation

An airship flight can last anywhere from a few hours to several days. During this time, the airship will experience various weather conditions, which in turn will affect the amount of solar radiation it receives and the amount of electricity it can produce. For a vehicle that relies entirely on solar power, it is essential to accurately predict the energy generated by the solar panels at any given moment. Different weather conditions, such as cloud coverage and the position of the sun in the sky, can influence solar radiation levels. Therefore, solar radiation must be computed using weather data based on specific locations, altitudes, and times.
While there exist methods to efficiently approximate the solar radiation that arrives at the earth for a clear sky, it is significantly harder to calculate the solar radiation for a cloudy sky. For an airship that is heavily dependent on the energy generated by solar panels, it is important to get the most accurate solar radiation data possible. Therefore, the new model will feature a more sophisticated approach to calculating the solar radiation that arrives at the airship. Instead of relying on estimates, this model will utilize ray tracing to determine the solar radiation received by the airship.

5.1 Calculation of the solar radiation

For the airship pathfinder to effectively determine a flight route, it is essential to calculate the solar radiation arriving at the airship for every location along that route. Flying a path with low sun exposure would result in underutilization of the solar panels, forcing the airship to rely on battery power for extended periods. The airship pathfinder software already features a basic approximation of the incoming solar radiation. This approximation, however, does not account for cloud cover, which significantly impacts the efficiency of solar panels. Currently, the Pathfinder only uses wind speed as weather data for route planning and expects every node on the route to have perfect solar radiation. This not only leaves it unable to reliably calculate the solar power that can be generated by the airship for a given route but leads to dramatically overestimating the power available to the airship. This overestimation can lead to the airship taking routes that are occluded by clouds for extended periods of time and, therefore, not generating any solar power, leaving the airship running on battery power or, even worse, running out of battery power. As the main part of a carbon-neutral airship is its ability to regenerate power by using the sun, it is necessary that this computation be done as accurately as possible. Therefore, the new model will use a more sophisticated approach to calculating the solar radiation that arrives at the airship. While this new approach requires more time to compute than the original model, various optimizations have been implemented to streamline the process. Additionally, users can deactivate this new model at any time and revert to the previous version using compiler flags, should they choose not to utilize it.

5.2 Reference model

The airship pathfinding software already features a model for the calculation of the solar radiation that arrives at the airship. This model is based on the works of [6] and provides a simplified approximation of incoming solar radiation. It enables the calculation of the solar radiation at a given height above ground for a given sza. The reference model approximates the incoming solar radiation by using the approximation for the air mass using the following formula by [6]:

$$M(z_t) \approx \frac{1.003198\cos(z_t) + 0.101632}{\cos^2(z_t) + 0.090560\cos(z_t) + 0.003198}$$

Here z_t is the sza. Afterwards the solar radiation is computed by:

$$s = G_{SC} \cdot 0.8^{M(z_t) \cdot \frac{h_{\text{atmos}} - h}{h_{\text{atmos}}}}$$

where G_{SC} is the solar constant, h_{atmos} is the height of the atmosphere, h is the height of the airship and $M(z_t)$ is the air mass. The 0.8 factor is used to account for the absorption of the atmosphere, which was chosen to be 20%. Although this formula provides a simple and quick method for approximating incoming solar radiation, it does not account for variations in solar radiation under different weather conditions. And as an airship is a vehicle that is heavily dependent on the weather conditions, this is a significant downside. Cloudy weather, in particular, can greatly reduce solar radiation; therefore, it is essential to develop a model that can accommodate these variations in solar power.

5.3 LibRadtran

As developing a complete atmospheric radiation model is extremely time-consuming, and testing its accuracy is even more so, it was decided to use an already existing model. Library for Radiative transfer (LibRadtran) was chosen as the radiation model as it allows customization of all required parameters, has extensive documentation, and is open source, which allows it to be recompiled on a target platform. This capability is particularly valuable when running the code on a High Performance Computing (HPC) cluster. LibRadtran is a sophisticated raytracer that calculates both solar and thermal radiation in the Earth's atmosphere [7]. It allows for more precise radiation data while simultaneously offering more configuration options than the old model. The Radiative transfer simulations component of LibRadtran called "uvspec" runs a 1-D Simulation for the solar power per m^2 for given input parameters. These input parameters include the day of the year, water and ice cloud coverage, terrain height, altitude of the simulated position, and the sza. While there are many more parameters that can be adjusted, these are especially relevant for the airship simulation.

5.4 Comparison of model results

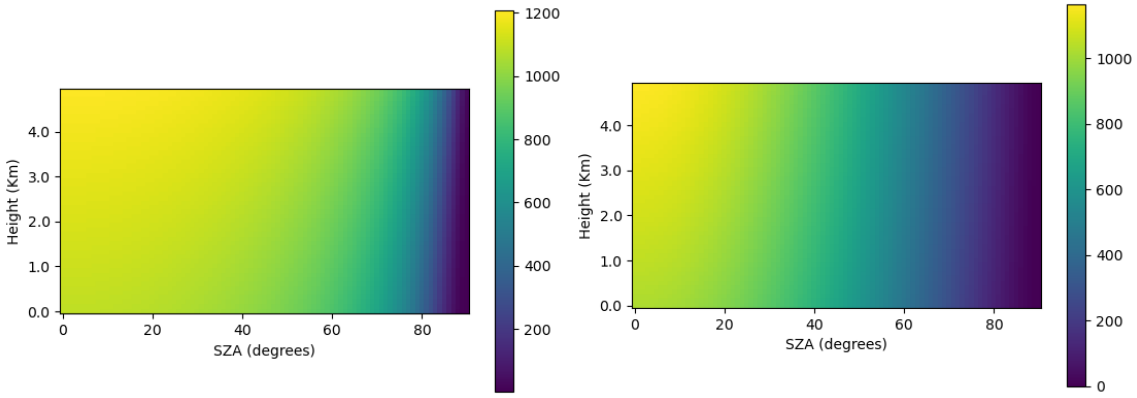

Figure 9: The calculated solar radiation per m^2 at a given height and sza. Left the old model and right the LibRadtran output. The LibRadtran version was calculated for a "day_of_the_year" of 170.

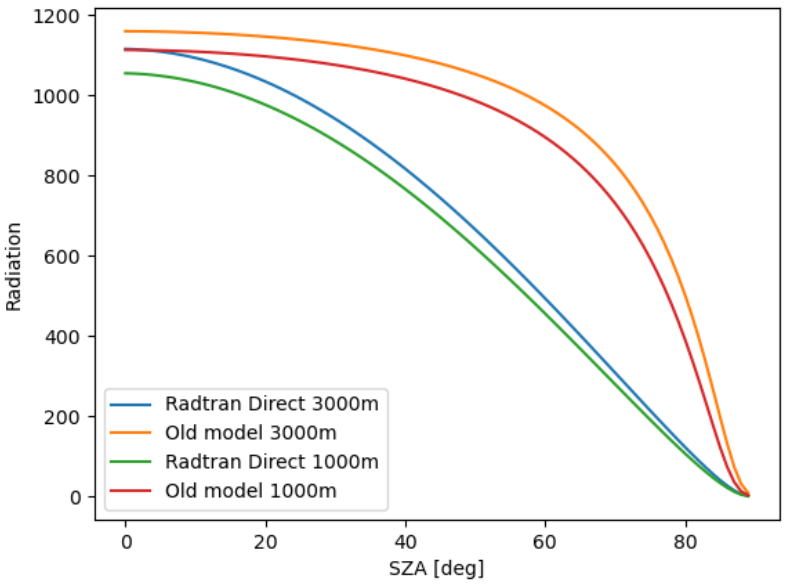

Figure 10: The calculated solar radiation per m^2 at the heights of 1 and 3 kilometers above ground for given szas. Plotted are the curves for the old model and the direct radiation calculated by the LibRadtran model. The LibRadtran version was calculated for a "day_of_the_year" of 170.

Before delving deeper into the functionality and configuration options of LibRadtran, it is interesting to see what the differences are when compared to the old model. As such, comparison calculations were done using no cloud coverage or other weather effects.
The heatmap plots of Figure 9 were generated by calculating the solar radiation for a given height and sza for both the old model and the LibRadtran model. Each pixel represents a simulation with specific parameters. By comparing these two figures, one can see that the old model had a tendency to overestimate the solar radiation by a small amount. The values are generally a tiny bit higher, and this increase in power extends way further into the higher sza values, before sharply dropping off at around $80\,\mathrm{deg}$. The LibRadtran model, on the other hand, has a more realistic curve that is more in line with what one would expect. This trend is further supported by the plot in Figure 10. It is important to note that, although the difference might not appear significant at first glance, the two relatively close lines represent the old model and the LibRadtran model rather than showing the differences between the two models for the same calculations. These plots show a significant difference between the old model and the LibRadtran model for higher sza values. While the new model features a more linear decrease in solar radiation with increasing sza, the old model stays relatively high and then features a sharp drop-off at around the higher sza values. One of the other advantages of using the LibRadtran model is that it is able to take into account the effect of water and ice clouds on the solar radiation. These are not shown in the comparison plots, as the old model does not have this functionality. For the LibRadtran model, the direct radiation was therefore calculated without cloud coverage.

5.5 Integrating LibRadtran into airship pathfinder software

As the airship pathfinder software is written in C++, it is necessary to be able to call the LibRadtran simulation from C++. Unfortunately, calling the LibRadtran model from another program is more complex than just importing it as a library. The LibRadtran software provides an executable called "uvspec". This executable serves as a wrapper for the core library and is intended to be run from the command line. It reads the input from a configuration file, passes it to different internal LibRadtran routines, and writes the output to another file. The problem is that the uvspec wrapper contains thousands of lines of code distributed over different files and is not easily modified and integrated into another program. Even if it were integrated into the airship pathfinder software, it would then be necessary to recompile the whole LibRadtran software upon building the airship pathfinder software. This would significantly complicate the build process since LibRadtran relies not only on a C++ compiler but also on FORTRAN, Python, and Lex compilers. Therefore, it was decided to call the uvspec program from the airship pathfinder software as an external program.

5.5.1 Calling the Radtran simulation from C++

In order to use the uvspec program in the airship pathfinder it has to be called from C++ as an external program. This can be achieved using the ‘popen‘ function, which is available in C/C++. Normally, the uvspec program would be called from the command line like this:

```
uvspec < input_file > output_file
```

This process requires both an input file and an output file. Where the input file represents the configuration of the simulation and the output file represents the results of the simulation. Creating separate input and output files for every call to the uvspec program can be excessively time-consuming, leading to the creation and deletion of tens of thousands of files during the runtime of the airship simulation software. Therefore, it is necessary to minimize the number of files that have to be created. For example, the need for an output file can be removed by using a pipe. This allows the output of the uvspec program to be directly read without needing to be written into a file first. Additionally, instead of providing an input file, the input can be passed directly through the command line using the bash ‘«‘ operator. As a result, creating input and output files for every uvspec call is no longer necessary.
The output that is written to the pipe has to be read and then parsed by the airship simulation software. To avoid any warnings or error messages from the uvspec program interfering with the parser, the pipe will only read from the program’s Standard Output (stdout). Meanwhile, the Standard Error (stderr) will be redirected to the airship simulation software’s standard output, allowing users to see any potential error messages without confusing the parser.
To prevent the stderr of thousands of uvspec calls from cluttering the output of the airship simulation software, the user can choose to redirect the stderr to a file by setting the provided "`silence_errors`" flag to true. This will internally append "`2>error.log`" to the uvspec call, which reroutes the stderr to a file called "error.log".

```cpp
std::string exec_r(const char* cmd) {
  const int buffer_size = 128;
  std::array<char, buffer_size> buffer;
  std::string result;
  std::unique_ptr<FILE, decltype(&pclose)> pipe(popen(cmd, "r"), pclose);
  if (!pipe) throw std::runtime_error("popen() failed!");
  while (fgets(buffer.data(), static_cast<int>(buffer.size()), pipe.get()) !=
↪  nullptr) {
      result += buffer.data();
  }
  return result;
}
```

Figure 11: The code that is used to call the external uvspec program from C++ and read the output.

5.5.2 Performance using the LibRadtran model

The uvspec program is notably more complex and slower than the previous model. Given this increased runtime, it is essential to determine how often the radiation simulation will be invoked by the airship pathfinder software.
It turns out that, for a route from Frankfurt to Mumbai, the Pathfinder software will call the radiation simulation roughly 120 million times. This is a staggering amount of calls, and it is not feasible to run the uvspec program 120 million times to simulate a single route. Interestingly, after more investigation, it turns out that the actual route-finder is not really calling the radiation simulation that many times.
The airship route-finder uses two algorithms to find the best route. The first is the so-called "ShortestPathFinder", which finds the shortest path between two points using the windspeed data provided by the MERRA-2 dataset[1]. Afterward, the "BatteryOptimizer" will get this shortest path and call the radiation simulation in order to check this path viability, battery status and calculates the engine power for every node of the route. As the "BatteryOptimizer" only works on the path provided by "ShortestPathFinder", it actually works on a significantly reduced subset of all nodes. On a route from Frankfurt to Mumbai, for example, the "ShortestPathFinder" will reduce the number of nodes from hundreds of thousands to roughly 60.

Still, nearly all 120 million calls to the radiation simulation are made by the "BatteryOptimizer," suggesting that many of these calls may be duplicates. When looking at the different input parameters for the radiation simulation, it turns out that the radiation values only depend on the airship's position and altitude and the time of day. The rotation of the airship does not matter, as the radiation simulation calculates only the radiation for a square meter plane at the position of the airship. This result must then be multiplied by the actual solar area affected by this radiation. Therefore, the radiation simulation will remain the same regardless of the orientation of the airship. Any potential weather data is also not really relevant, as it changes depending on the time the airship visits the position. However, as we already have the time of day as an input parameter, calculating the same time of day again will not change the weather data. This means that the actual radiation simulation is really only dependent on two input parameters: the position and the time of day.
As there are only two input parameters, one of which is the position, and we know there are only 60 discrete path positions given by the "ShortestPathFinder" algorithm, there has to be a significant number of duplicate calls to the radiation simulation to justify the 120 million calls. This implies that the battery optimizer is making multiple calls to the radiation simulation for the same input parameters, indicating that these values should be cached to improve efficiency.

Figure 12: All positions that are checked for radiation values on a route from Frankfurt to Mumbai

To better understand the radiation accesses made by the airship pathfinder software, the positions where radiation values are checked on a route from Frankfurt to Mumbai are displayed in Figure 12. There, it is evident that most checked positions lie on a clear path with only a few deviations. As a result, implementing a sophisticated caching mechanism is likely to significantly reduce the number of calls to the radiation simulation without compromising the accuracy of the results. To be able to write an appropriate caching mechanism it is first necessary to find out how the distinct calls to the radiation simulation look like and how many of them are actually duplicates.

5.5.3 Caching the radiation values

To gain a better understanding of the radiation checks conducted by the Airship Pathfinder software, the positions where radiation values are checked on a route from Frankfurt to Mumbai are displayed in Figure 12. There, it is evident that most checked positions lie on a clear path with only a few deviations. This means that all hundred million radiation checks occur near positions on this line, resulting in a significant amount of redundant calculations. As a result, implementing a sophisticated caching mechanism is likely to significantly reduce the number of calls to the radiation simulation without compromising the accuracy of

the results. To be able to write an appropriate caching mechanism, it is first necessary to find out how the distinct calls to the radiation simulation look like and how many of them are actually duplicates.

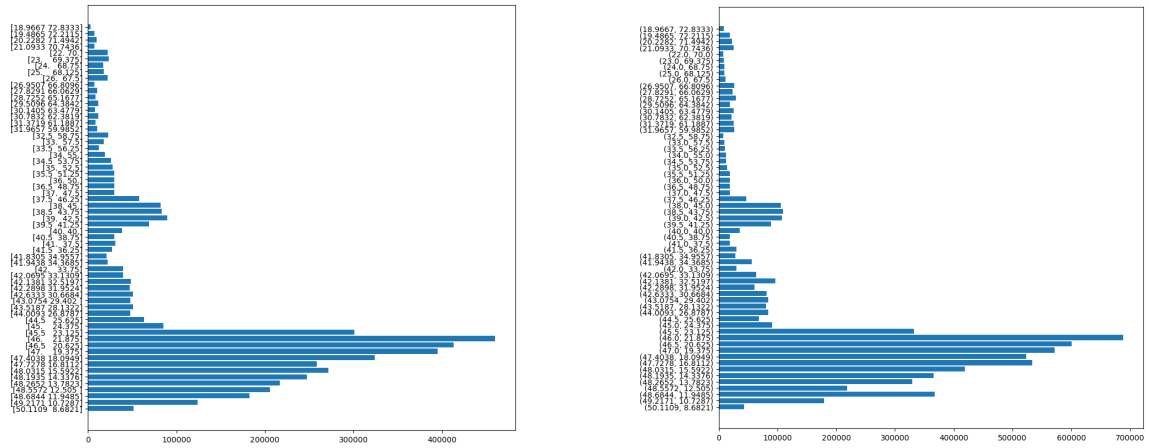

Figure 13: Subset of the positions that are checked for radiation values on a route from Frankfurt to Mumbai and the number of times they are checked. Only the positions that are part of the final route are shown. Left: Only count accesses on the exact path. Right: Count all accesses and add them to the nearest node on the path without counting accesses on the path itself. This shows the number of accesses that are not on the path and cannot be predicted.

As can be seen in Figure 12, the BatteryOptimizer checks numerous positions for radiation values, most following a straight path with only a few deviations. This suggests promising behavior for the caching of the radiation values.
If one then looks at the number of accesses for each position in Figure 13, one can see that this number is relatively high and can even reach values multiples of 10000. On the right side image of Figure 13, one can see that the number of accesses not directly on the path but slightly next to it is also substantial.
Due to the limited resolution of the MERRA-2 dataset, it is feasible to route several different positions to the same cache entry if they are sufficiently close. If a node shares the same MERRA-2 weather data as a neighboring position, there is no need to recalculate the radiation values for that node. That means that the cache implementation has to be able to decide if a value is close enough to a cached value to be used instead of recalculating it or if it should create a new cache entry. Such a solution reduces the number of radiation simulations that have to be run significantly.
Sadly, this approach leads to another problem. Even if the new position is near a cached neighboring position, the fact that it was close enough but not the same means that the time of day for this new position will most likely differ from this neighboring position. As time of day is a huge factor in radiation calculations, this means that the radiation values for the new position have to be recalculated anyway, defeating the purpose of the nearest position cache optimization. While this problem cannot entirely be resolved, it can be mitigated drastically. Since the radiation values are only dependent on position and time of day, the cache can be precalculated for multiple different "times of day" for a single position. Thus, every time the cache calculates radiation values for a new position, it computes those values for several different times of day as well.
These calculations have the advantage that they can trivially be parallelized and, therefore, speed up the cache-building process. Although the total amount of radiation calculations is not drastically reduced compared to a more trivial caching system, the actual runtime is far superior. While calculating the radiation values for the whole day seems excessive, it actually makes little difference in time if a 64-core CPU calculates two or 64 radiation simulations in parallel.
The resulting cache then allows for trivial access to the radiation values for any position and time of day. Once the radiation simulation is able to calculate multiple routes in parallel, this behavior should probably be changed to one that only calculates the radiation values for the current time of day and the immediately surrounding times. However, this would have to be tested first. The resulting cache can now significantly reduce the number of radiation simulations that have to be run. Instead of running all 120 million radiation simulations, the cache will only have to calculate the radiation values for the 60 nodes in the final route, plus some deviating alternative parts. It will perform n number of radiation simulations for each node, where n is the number of different times of day that are calculated for each node. In its current configuration, the

cache will calculate the radiation values for 36 distinct "times of day" for each node. This results in a total of $60 * 36 = 2160$ radiation simulations that have to be run. This is a significant reduction compared to the 120 million simulations that would have to be run without the cache. It may further be drastically reduced by calculating only the radiation values for the current time of day and the immediately surrounding times. However, as the cache is already able to calculate the radiation values for the whole day in a reasonable amount of time, and this work can be parallelized efficiently, this is not a priority at the moment.

5.5.4 Parallelizing the cache building process

If a solar radiation value is needed for a position that is not already in the cache, the cache has to be updated. The battery optimizer may try to request multiple different daytimes for the same or similar positions. To avoid recalculating the cache for each of these requests, the LibRadtran simulation will be computed for multiple instances during the day and then interpolated between them. These calculations are done in parallel using Open Multi-Processing (OMP) to speed up the cache-building process. In order for this to work, not only has the cache to be thread-safe but also the call to LibRadtrans uvspec executable. Fortunately, the input and output files for the uvspec program can be skipped by writing the input directly into the command line and reading the output from the stdout via a pipe. This allows for multiple instances of the uvspec program to be run in parallel without the need for multiple input and output files. Sadly, the extra cloud data provided by the MERRA-2 dataset cannot be provided via the command line and has to be written into a file. In the current version of the airship Pathfinder, however, the weather data stays constant for the whole trip. This means that only one file has to be written and read by all instances of the uvspec program for every position. Once multiple times of days for datasets are supported in the pathfinder software, the cloud data will have to be written into multiple files. This should not be a problem as the cloud data files for LibRadtran are each only a few lines in size. And the currently used MERRA-2 dataset only provides eight timesteps per day, meaning that these time-dependent files will also be very limited.

6 Simulation of solar radiation with weather data

The new radiation simulation based on the LibRadtran model is providing more accurate results than the old model. Neither model, however, does take into account the effect that clouds will have on the solar radiation reaching the airship. The influence of clouds on solar power generation is well-known and can be significant. For an airship that relies entirely on electricity, having enough electrical power is crucial to stay on course. Therefore, it is important to have a model that can take into account the effect that clouds will have on the solar irradiance that reaches the airship.

6.1 Combining the weather data with the radiation simulation

6.1.1 Weather data (Merra-2)

The MERRA-2 dataset provides global weather data, released by NASA and available for free use. Currently, the airship simulation already uses the MERRA-2 dataset for the wind speeds used by the airship pathfinder. In addition to wind data, this dataset contains a variety of other information that may be useful for radiation simulations. Since the resolution of this dataset is also the resolution in which the whole airship path is calculated, it would be perfect to also use this dataset for the radiation simulation. The dataset is stored in the NetCDF4 file format, which allows for efficient data storage and access.

Key	Description	Unit
EPV	ertels potential vorticity	$K\ m^2 s^{-1} kg^{-1}$
H	edge heights	m
lat	latitude	deg
lon	longitude	deg
lev	vertical level	hPa
O3	ozone mass mixing ratio	$kg\ kg^{-1}$
OMEGA	vertical pressure velocity	$Pa\ s^{-1}$
PHIS	surface geopotential height	$m^2\ s^{-2}$
PS	surface pressure	Pa
QI	mass fraction of cloud ice water	$kg\ kg^{-1}$
QL	mass fraction of cloud liquid water	$kg\ kg^{-1}$
QV	specific humidity	$kg\ kg^{-1}$
RH	relative humidity after moist	1
SLP	sea level pressure	Pa
T	air temperature	K
time	time	s
U	eastward wind	$m\ s^{-1}$
V	northward wind	$m\ s^{-1}$

Figure 14: The different kind of data that is available in the MERRA-2 dataset.

The airship route finder currently utilizes wind data, specifically the U and V components (see Figure 14), to predict the speed of the airship and the direction it will take. The goal is now to use the cloud coverage data QI and QL to calculate the solar radiation in cloudy conditions. The cloud coverage data is given as the mass fraction of cloud ice water and cloud liquid water. While this is a format used by many weather models, it is not the format that is expected by the LibRadtran model. The LibRadtran model expects the cloud data to be in a more optically significant format. This format being the liquid water content (LWC) and the effective droplet radius R_{eff}. In order to convert the cloud data from the MERRA-2 dataset to the format that is expected by the LibRadtran model, it is necessary to use other MERRA-2 data points than the cloud coverage data.
As the MERRA-2 dataset is a global dataset based on real-world data, it is not possible to get the exact cloud shape and position data for a given area. The resolution of the dataset is also in the order of kilometers and not meters. While this sounds detrimental to an accurate radiation simulation, it is actually not a big problem. It is important to remember that we are not interested in the exact radiation that reaches the airship at a certain timestep but rather the average radiation that reaches the airship over a certain period of time. Therefore, the information about the average cloud coverage over a certain area is enough to calculate the

average radiation that reaches the airship during its travel through this area. An ultra-high-resolution dataset would only increase the runtime of the simulation without providing any additional benefit.

6.1.2 LibRadtran file format for weather data

```
# z      LWC      R_eff
# (km)   (g/m3)   (um)
  5.000 0        0
```

LibRadtran expects the cloud data to be in a specific format. It needs the altitude of the cloud in kilometers, the liquid water content (LWC) in g/m^3, and the effective droplet radius R_{eff} in μm. The altitude in kilometers can easily be calculated from the pressure in hPa provided in MERRA-2 under the lev key by using the barometric formula.

$$h = \frac{T_0}{L_0}\left(1 - \left(\frac{P}{P_0}\right)^{1/5.255}\right)$$

This can easily be converted to C++ code:

```cpp
// conversion constants defined in file radtran_params.hpp
namespace radtran_params{
  // reference constants for h_0 = 0
  constexpr static double T_0 = 288.15; // K
  constexpr static double P_0 = 1013.25; // hPa
  constexpr static double L_0 = 0.0065; // K/m

}
double hPa_to_meters(double height_in_hPa){
  if (height_in_hPa < 0){
      throw std::runtime_error("...");
      // or return std::numeric_limits<double>::quiet_NaN();
  }
  return radtran_params::T_0/radtran_params::L_0 * (1 - std::pow(height_in_hPa /
↪  radtran_params::P_0, 1/5.255));
}
```

Unfortunately, the LWC values cannot be obtained that easily. When looking at the datatypes that are available in the MERRA-2 dataset seen in Figure 14, one can see that the cloud data is not available in this format. The cloud data is instead given as the mass fraction of cloud ice water and cloud liquid water. This means it is necessary to convert the cloud data from the MERRA-2 dataset to the format that is expected by LibRadtran. The conversion from QI and QL, which are provided by MERRA-2, to LWC needed by a lot of optical solvers like LibRadtran can be done using the following formulas [8].

$$LWC = q_{\text{liquid}} \cdot \frac{10^2 P}{R_{\text{moist}} T}$$

here q_{liquid} is the mass fraction of cloud liquid water provided under the key QL in the MERRA-2 dataset. This leaves P as the pressure in hPa, T as the temperature in K, and R_{moist} as the gas constant for moist air as unknowns. P and T are provided by the MERRA-2 dataset under the keys lev and T respectively. This leaves the gas constant for moist air R_{moist} as the only unknown. It can be calculated by:

$$R_{\text{moist}} = R_{\text{dry}}(1 + \frac{1-\varepsilon}{\varepsilon} q_{h2o})$$

here R_{dry} is the gas constant for dry air, q_{h2o} is the specific humidity and the coefficient ε. The specific humidity q_{h2o} is provided by the MERRA-2 dataset under the key QV. This leaves the coefficient ε and R_{dry} as the only unknowns. Both can easily be calculated by:

$$\varepsilon = \frac{M_{h2o}}{M_{\text{dry}}}$$

and

$$R_{\text{dry}} = \frac{R}{M_{\text{dry}}}$$

with M_{h2o} being the mean molar mass of water and M_{dry} being the mean molar mass of dry air and R being the universal gas constant.

$$M_{h2o} = 18.01528 \ \frac{g}{mol}$$
$$M_{\text{dry}} = 28.9644 \ \frac{g}{mol}$$
$$R = 8.31446261815324 \ \frac{J}{mol \ K}$$

written down in code this looks like this:

```cpp
// conversion constants defined in file radtran_params.hpp
namespace radtran_params{
  constexpr static double M_h2o = 18.01528; // g/mol (mean molar mass)
  constexpr static double M_dry = 28.9644;  // g/mol (mean molar mass)
  constexpr static double R = 8.31446261815324; // J/(mol K) (ideal gas constant)
  constexpr static double M_epsilon = M_h2o / M_dry;
  constexpr static double R_dry = R / M_dry;
}

//----- conversion functions -----

// returns the moist air gas constant (in Jg-1 K-1)
double R_moist(double q_h2o){
    return radtran_params::R_dry * (1 + (1 -
↪   radtran_params::M_epsilon)/radtran_params::M_epsilon * q_h2o);
}

/**
  * @brief converts [kg/kg] to [g/m^3]
  *
  * @param q_ice mass fraction of cloud liquid or ice water content [kg/kg]
  * @param P     pressure [hPa]
  * @param q_h2o specific humidity [kg/kg]
  * @param T     temp. [K]
  * @return double liquid water content (LWC) [g/m^3]
  */
double convert(double q_ice, double P, double q_h2o, double T){
    return q_ice * (100 * P)/ (R_moist(q_h2o) * T);
}
```

Therefore, in order to calculate the liquid water content LWC from the MERRA-2 dataset, the QL, P, QV, and T values are needed. A similar approach can be used to determine the cloud ice water content IWC. In this case, instead of the mass fraction of cloud liquid water q_{liquid}, which is found in MERRA-2 as QL, the mass fraction of cloud ice water QI is used. All other calculations remain the same.
The last remaining value that is needed for the LibRadtran input file is the effective droplet radius R_{eff}. This value is set to a constant value of $10\mu m$ for water clouds and $30\mu m$ for ice clouds.

6.1.3 Extracting and reading the data from the Merra-2 dataset

The MERRA-2 dataset is provided in the NetCDF4 format. As reading the NetCDF4 format is not trivial in C++ and not all of the data is needed, the required data will be saved in a more accessible format first. Currently, such formats have been written for the already used wind data in the airship Pathfinder software. The data for the LibRadtran model, however, will need to be structured differently. Not only does it need more different datatypes, QL, QI, T, and QV, but it also needs the data to be structured in a different way. For a given position and time, the LibRadtran model requires information from every of the 42 cloud layers of the MERRA-2 model. The current MERRA-2 reader used for the wind data only reads the data for a position and a time. If one were to apply this method to the cloud data it would result in 42 data searches per

MERRA-2 key entry. With QL, QI, T, and QV, this would result in $4 \cdot 42 = 210$ data searches per position and time. This would unnecessarily increase the runtime of the Airship Pathfinder software. Using a more practical data format would allow to access all this data easily. The data will be saved in a CSV file with the following format:

```
lat lon QL_(level=0) QL_(level=1) ...
0   0   0            0            ...
```

The file contains the data of all QL levels for a given latitude and longitude in a single line. That way, all values for different heights are on the same line and can be easily read. This reduces the number of data searches from 42 to 1 per MERRA-2 key. Additionally, separate CSV files will be generated for QI, T, and QV. These files are created by an external Python script once before the airship pathfinder software is run. The airship pathfinder software will then read these files and cache them in memory for faster access. In its current configuration, these files are each around 100MB in size. Once the resolution of the underlying weather data is increased and the size of these files becomes too large, they can easily be decreased in size by limiting the number of decimal places or by using binary encoding. Alternatively, these files can easily be compressed using a compression algorithm like gzip. Note that the P-key values are not saved to an extra file because they can be calculated via the level number and the corresponding conversion provided in the MERRA-2 dataset.

6.1.4 Using the weather data in the airship simulation

In order to effectively use the weather data in the airship simulation, a new class called "Radtran_Cloud_Reader" has been implemented. Upon its creation, this class will read the contents of all the CSV files that were created by the Python script. These files contain all the cloud data needed for the radiation simulation at various latitudes and longitudes. It will then save the data into a ‘std::vector‘ of type "Radtran_Cloud_Reader::Cloud_Entry". Each "Cloud_Entry" consists of latitude, longitude, and a ‘std::vector‘ of cloud data for QL, QI, T, and QV. These vectors each have one entry for every height level available in the MERRA-2 dataset.

```
class Cloud_Entry{
private:
  ...
public:
  float latitude;
  float longitude;
  std::vector<double> QL; // mass fraction of cloud liquid water content [kg/kg]
  std::vector<double> QI; // mass fraction of cloud ice water content [kg/kg]
  std::vector<double> QV; // specific humidity [kg/kg]
  std::vector<double> T;  // temperature [K]

  ...
};
```

Figure 15: The Cloud_Entry class that is used to store the cloud data for a given latitude and longitude.

The "Radtran_Cloud_Reader" class includes a method that will find the correct cloud data for a given latitude and longitude and write it into a file that can be read by the LibRadtran program. If there is no exact match for the given latitude and longitude, the closest available match will be used instead. Once the data is prepared, the LibRadtran simulation is executed using this file as input, allowing for the calculation of radiation values based on the provided cloud data. This allows the weather data to be included seamlessly in the already presented Radtran cache.

6.2 Calculating radiation using weather data

6.2.1 Radtran with weather data

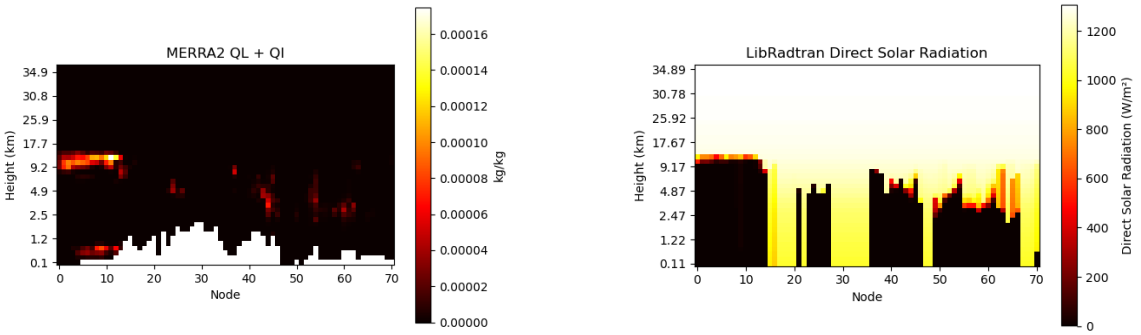

Figure 16: The two graphs feature as y-axis the height in kilometers and as x-axis a position on a route from Frankfurt to Mumbai. Left: The cloud coverage from the MERRA-2 dataset. Right: The direct radiation values calculated by libradtran for the MERRA-2 cloud data. For comparability the solar zenith angle was set to 0 for the whole route.

To better understand how weather data influences the irradiance values reaching the airship, it is important to visualize both the cloud data and the corresponding radiation values.
For example, Figure 16 shows the cloud coverage for a route from Frankfurt to Mumbai. In the left plot, the combined values for the cloud ice water and cloud liquid water content are displayed for every node along the airship's path. Using this data, the LibRadtran model is able to calculate the direct radiation values for the given cloud coverage at different heights, as shown in the right plot of Figure 16.
The values are at their highest at the top of the atmosphere and decrease with the altitude, even if no clouds are present. However, if the internal raytracer of the LibRadtran model hits a cloud, the direct radiation values will decrease significantly, reaching zero almost instantly. It is relevant to note here that a commercial airship will typically fly at a maximum altitude of around 3.5km. While it is possible for an airship to fly substantially higher, this usually means less payload capacity, which defeats the purpose of using airships for transport. At such a low altitude, there are still a lot of clouds above the airship blocking the sun, resulting in no direct solar radiation reaching the ship. Therefore, when just looking at the graph, one can see that the radiation values provided must be zero for nearly all points of the route. This is not useful as the airship will still receive radiation even if it is under cloud coverage. In order to still get a realistic radiation value for the airship, the diffuse radiation values have to be used.

6.2.2 Diffuse radiation

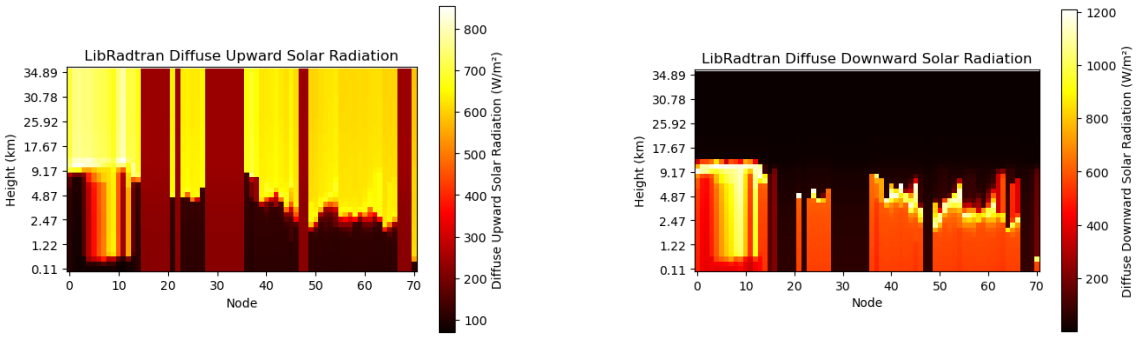

Figure 17: The diffuse radiation values for the same path nodes as in Figure 16. Left the upward diffuse radiation and right the downward diffuse radiation.

The diffuse radiation values calculated by the LibRadtran model provide irradiance data for locations that are not directly hit by the sun. LibRadtran uses two types of diffusive radiations, the downward facing and the

upward facing one. They are both defined as the integral of the radiances over their respective hemisphere. Using these definitions, it is also possible to get radiation values for locations where the direct radiation values approach zero. This includes locations fully occluded by clouds, as well as locations that are under no cloud coverage at all. For example, even without clouds, the earth's albedo will reflect some of the radiation back into the atmosphere and hence generate upward radiation. This radiation is calculated as the diffuse upward radiation by LibRadtran.
This is especially useful for the airship simulation as most airships have a travel altitude of around 3.5km and are, therefore, under cloud coverage for a significant amount of the time. In the left plot of Figure 16, one can see the cloud information of the MERRA-2 dataset for a route from Frankfurt to Mumbai. On this route, the airship will be under cloud coverage most of the time. If one only uses the direct radtran radiation values, like in the right plot of Figure 16, the radiation values will be zero once the raytracer crosses a cloud. This is not realistic as the radiation values should only be reduced by the cloud coverage and not be zero. Therefore, the diffuse radiation values have to be used also. The upward and downward diffuse radiation values can be seen in Figure 17.
Upon closer inspection, one can see that the downward diffuse radiation values often reach a significant portion of the direct radiation values. For example, at nodes 5 to 10, the downward diffuse radiation values are roughly the same as the expected direct radiation values without cloud coverage. This means that the airship will still receive a significant amount of radiation even if it is under cloud coverage.
The key difference between diffuse and direct radiation values lies in their origin; diffuse radiation can come from any direction, not just from the sun. Consequently, solar area calculations must be adjusted to account for the diffuse radiation values.

6.2.3 Solar area calculation with diffuse radiation

In order to get the correct solar radiation value for the solar panels, it is necessary to include the diffuse radiation values in the calculations. These values are not dependent on the direction the airship is facing and are considered to be coming from all directions. Both the downward and upward diffusive irradiances are defined as the integral of the radiance over their respective hemisphere.

$$E = \int L(\mu, \varphi) d\mu d\varphi$$

with

$$\mu = \cos(\theta) \quad \text{and} \quad \varphi = \text{azimuthal angle}$$

A solar panel is theoretically able to receive radiation from all directions that are not occluded by itself. If one considers the solar panel to be a simple 2D plane with a normal vector $\vec{n}$, one can calculate the radiation received by integrating the radiance over the whole hemisphere defined by the normal vector. I.e., one must integrate over all possible directions $\vec{d}$ that feature an angle $\alpha < 90\,\text{deg}$ with the normal vector.

$$\alpha = \angle(\vec{n}, \vec{d})$$

The uvspec program provided by LibRadtran is able to calculate the radiance values for a given direction. However, it provides no way of calculating the integral over the whole hemisphere for given angles. It only calculates the direct irradiance and the diffuse upwards and downwards irradiance. Consequently, users need to calculate irradiance for other directions independently. Since each irradiance calculation requires radiances over the entire hemisphere, this can lead to a considerable increase in computation and complexity. Obtaining the radiance values for all these directions from the LibRadtran model and numerically integrating over them is not trivial and is further explored in section 9. As a result, the following section will focus solely on using the provided upward and downward diffuse radiation values, and it will interpolate between them instead of addressing special direction irradiances.

6.2.4 Diffuse radiation on the underside of the airship

While the presence of high diffuse downward radiation is not surprising—since it is evident that even with cloud cover and no directly visible sun, light still reaches the Earth's surface—the existence of relatively high upward diffuse radiation is unexpected. One might typically expect such behavior when the ground is covered in snow, as snow features an albedo of nearly 1 and, therefore, reflects most of the radiation back

into the atmosphere. Nevertheless, when looking at the left plot of Figure 17, one can clearly see that the upward diffuse radiation values are sometimes surprisingly high. At nodes 9 and 10, for example, the upward diffuse radiation values are around 700 $W/m2$. This is roughly 50% of the expected direct radiation values on a clear day. This means that the airship has the potential to receive a significant amount of radiation from beneath. Therefore, installing solar panels on the underside of the airship could make sense. To see if this is feasible, the radiation values for the underside of the airship have to be calculated in more detail and for different cloud coverages.

```
# z         LWC           R_eff
# [km]      [g/m^3]       [um]
0.988664    0             10
0.762093    0.000279912   10
0.540427    0.00046644    10
0.323435    4.36038e-07   10
0.110903    0             10
```

Figure 18: A sample LibRadtran cloud data input file featuring areas of increased LWC at low altitudes. Taken by using only the liquid water values of node 7 shown in Figure 16.

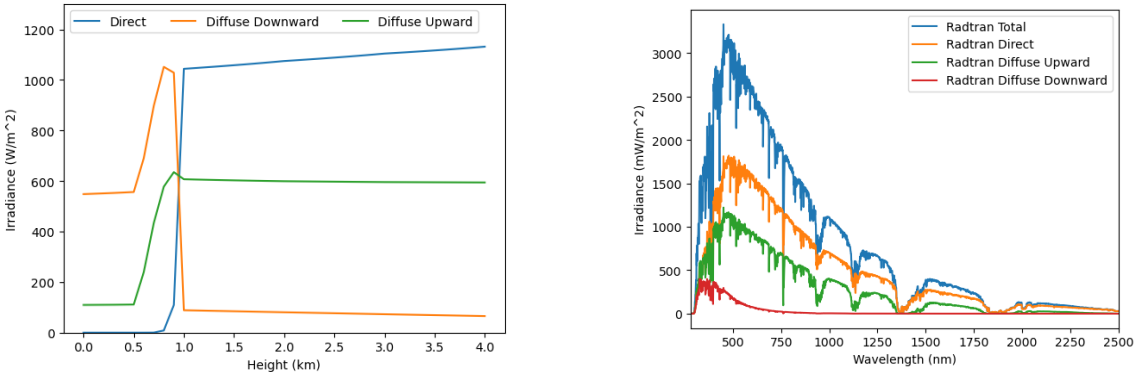

Figure 19: Radiation plots for the LibRadtran cloud input file from Figure 18. Left: The different radiaton values over different heights. Right: The different radiation values over all wavelengths at a height of $3km$.

In order to further investigate the potential of solar panels on the underside of the airship, a sample cloud data input file for the LibRadtran model has been created. This file features a cloud layer situated between 0 and 1 km with a relatively low liquid water content. The radiation values for this cloud data can be seen in Figure 18. By using these data values as input for the LibRadtran simulation, it is possible to visualize the varying radiation levels at different heights. The left plot of Figure 17 clearly illustrates the existence of a cloud layer at around 1 km, where the direct radiation values rapidly decrease to zero starting at this altitude. Additionally, both downward and upward diffuse radiation values experience a significant increase beginning at approximately 0.5 km. At this altitude, the cloud's liquid water content starts to play a crucial role as an optical factor. The LibRadtran model is also able to calculate the resulting irradiance on a per-frequency basis. This can be seen in the right plot of Figure 17 for a height of $3km$. Here, one can not only see that the diffusive upward radiation values reach values that have a significant portion of the combined incoming radiation but also that the resulting spectrum is slightly shifted to smaller wavelengths. This becomes even more apparent when looking at the diffuse downward radiation. The substantial upward diffuse radiation at this height indicates a significant potential for solar panels on the underside of airships.

7 Resulting calculations of the routefinder

All up until now discussed calculations have been implemented and included in the airship pathfinder software. The software is now able to use the increased accuracy of the LibRadtran model to better calculate the radiation values for a given route. This should allow for a more realistic simulation of the airship Pathfinder. Additionally, all existing references to the solar energy calculation have been replaced by the new LibRadtran model, along with the updated solar area calculation.

7.1 Example flight from London to New York

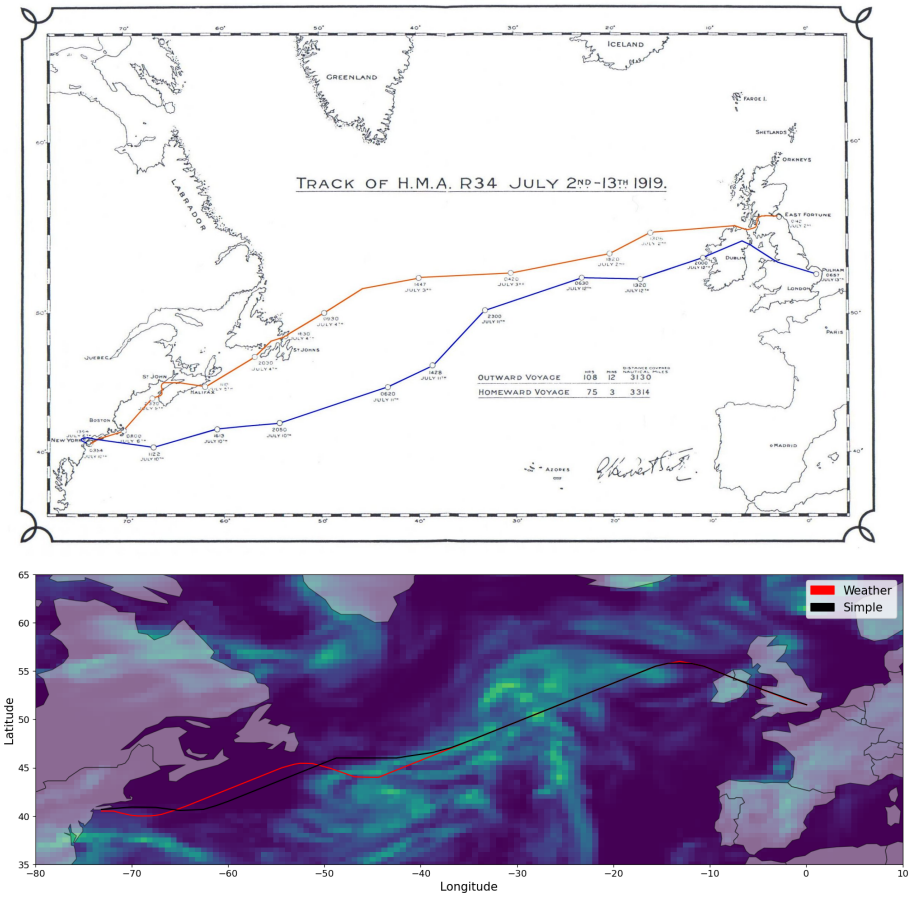

Figure 20: Top: The flightpath of the first transatlantic airship flight. [4]
Bottom: The flight path generated by the airship pathfinder software for a route from London to New York with a max height of $3500m$.

For this example, the flight route from London to New York has been chosen. This route is particularly relevant as it has been traversed by airships in the past, making it an ideal test case for the airship Pathfinder software. In the top image of Figure 20, one can see the original flight path of the first transatlantic airship flight in 1919, conducted by the British airship R34 [4]. The bottom image shows the flight path generated by the airship Pathfinder software for a route from London to New York with a maximum flight altitude of 3500 meters. There, the Simple route describes the route calculated using the old radiation system, and the Weather route describes the route calculated using the new LibRadtran model. Both simulations utilize the updated solar area calculation.
As can be seen, the difference in the flight path for this flight is not very significant. This may be because even with weather data included, the irradiance that reaches the airship is still high enough to allow for a direct flight. Since the current route already optimizes battery power storage and engine power utilization, pursuing a more complex route that could potentially enhance solar power generation may not be necessary.

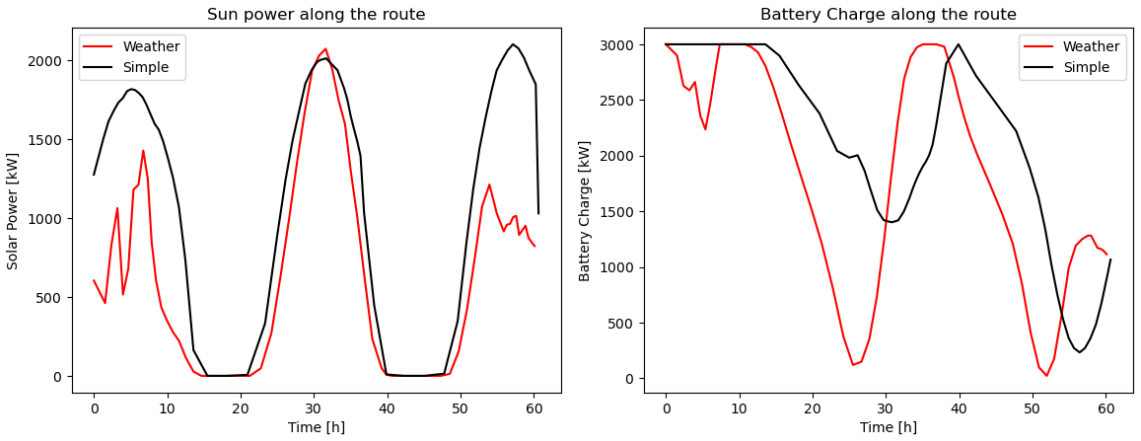

Figure 21: The solar power generated by the airship for a route from London to New York with a max height of $3500m$. The Simple version uses the original radiation approximation to calculate the solar power and the Weather version uses the new LibRadtran model.

Figure 21 shows the solar power generated by the airship along the two routes depicted in Figure 20. The changes made by the new LibRadtran based radiation model are clearly visible. In the first few hours of the flight, the solar power generated by the airship fluctuates strongly because of the newly included weather data. While during the middle of the flight, the new and old radiation models generate nearly the same amount of solar power. Towards the end, there is again a significant difference between the two models. This difference is even more pronounced in the battery power plot. It can be seen that with the new model the airship is relying heavily on the fully charged battery at the beginning of the flight. Here, the improved radiation model clearly had an effect on the overall solar power generated by the airship. Although that difference may not have been substantial for this flight and, therefore, did not result in any significant changes in the flight path, it is still a good example of how the new radiation model can affect the airship pathfinder software. Another flight that could be interesting to look at is the same flight but in the winter. Given that the airship's solar power generation will be considerably lower during this season, the new radiation model may have a more pronounced effect on the flight path. Details of such a flight can be found later in Section 7.5.

7.2 Cloud dependent flight altitude

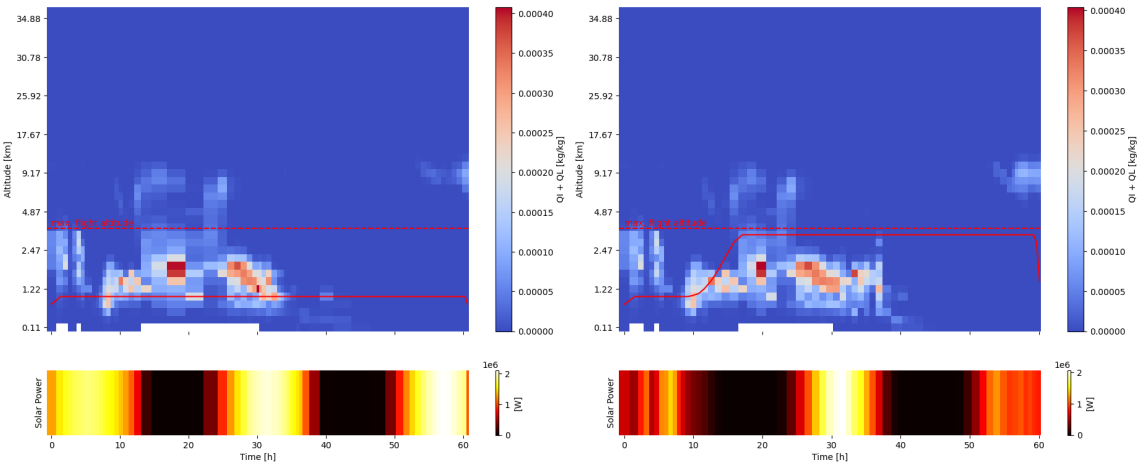

Figure 22: Radiation cache accesses for a route from London to New York in the summer with a max height of $3500m$.

While the new resulting route for the summer flight from London to New York is not very different from the original route when considering the position of the ship, there is a huge difference when considering the flight altitude. The hourly flight height is illustrated in Figure 22. These plots also contain the weather info for their respective routes. Although the cloud data is largely consistent, as both routes are fairly similar, there are some slight variations. The new model spends more time over the clouds than the old model, and the time spent under the clouds is smaller. This makes sense, as being under a cloud can significantly reduce the solar power generated by the airship. In general, the flight altitude of the new model corresponds to the cloud data relatively well. After starting under the first cloud layer, the airship quickly rises above the clouds and stays there for the remainder of the flight. In contrast, the old model had no idea of the actual weather and, therefore, chose to stay at the same height for the duration of the flight.
In the Solar Power plot beneath the main cloud data plot, one can see the solar power generated by the airship. The old model did not care about the weather and, therefore, features a relatively clear solar power output that is only interrupted by night. The new model, on the other hand, features a more realistic solar power output that fluctuates depending on cloud coverage. Each flight starts with a fully charged battery on board, which might explain why the new version does not feel the need to ascend quickly above the first cloud layer.

7.3 Theoretical maximal radiation during the flight

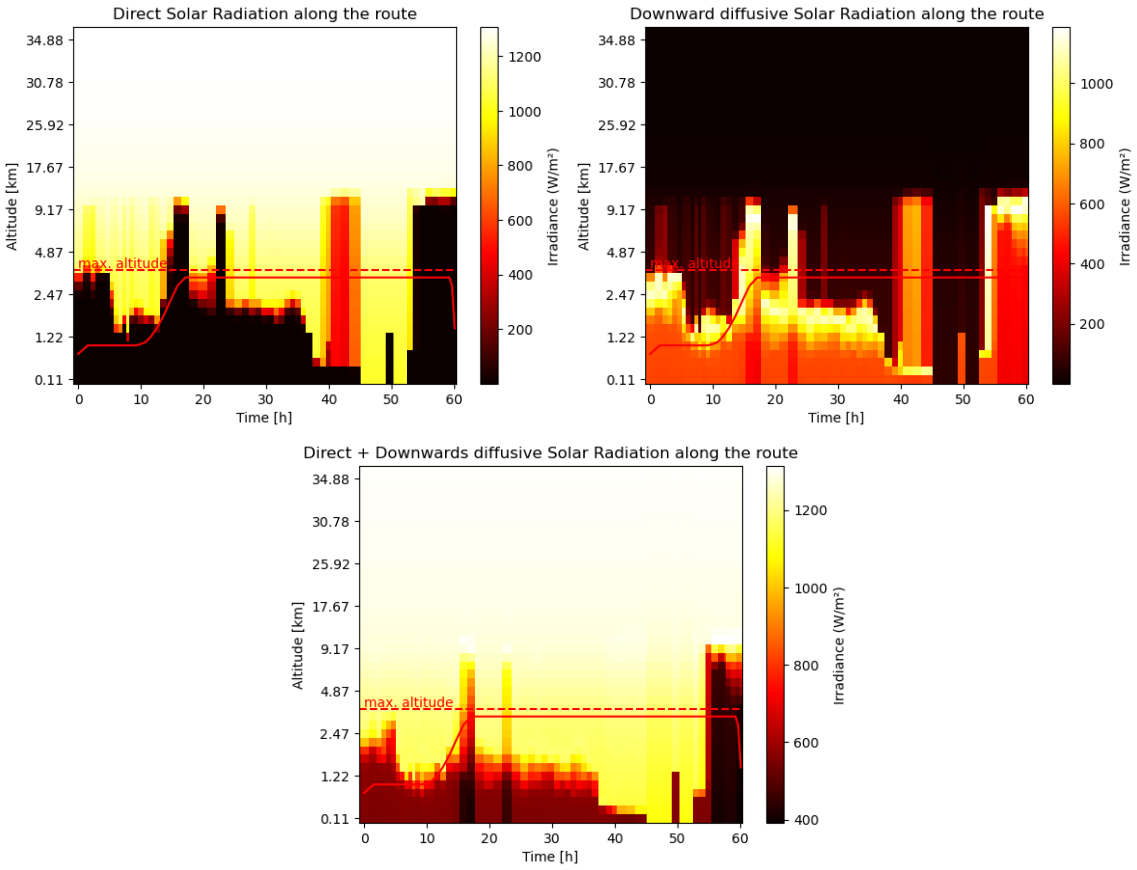

Figure 23: Radiation cache accesses for a route from New York to London in the winter with a max height of $3500m$. For all the radiation simulations the sza has been set to 0.

The radiation values for the route from London to New York in the summer have been calculated. These radiation values are based on the cloud data from the MERRA-2 dataset for the given route shown in Figure 22 and have been calculated using the LibRadtran model for every node on the route and every available data height. The radiation values for the route from London to New York in the summer can be seen in Figure 23. It is important to note that the actual time of day during the flight is not considered. Instead, it

is assumed that it is midday at every node. This allows one to see the potential of solar energy generation throughout the route. As the direct irradiance values stop once they hit a cloud layer, the diffusive downward radiation values have to be used. These values are not as high as the direct radiation values but still provide a significant amount of radiation. By adding both together, one gets the total radiation values that reach the top side of the airship. This plot can be seen in the bottom row of Figure 23. While the flight altitude looked rather weird in the direct radiation plot, as the first 20 hours the airship was flying mainly through clouds and therefore did not seem to get much light, now when looking at the combined graph, one can see that the airship was actually flying through a significant amount of solar radiation. For the first few hours until it reached its maximum flight altitude, it carefully flew above the clouds and increased its flying altitude as the cloud height increased. This lasted up until around 16 hours into the flight when suddenly the cloud height far exceeded the maximum flight altitude of the airship.
There are three parts of the route that experience less than ideal light conditions when looking at the combined radiation values. The first is right at the beginning of the flight when the airship is still beneath the first cloud layer. This cloud layer cannot be avoided as the start position parameters are fixed. When looking at the cloud data plot in Figure 20, one can see that the airship could have prevented much of the cloud occlusion in the first few hours by flying a bit more to the south instead of the north. The Pathfinder probably decided against that, as going northward is the quicker route. This can be seen as the "Simple" calculation, not using any weather data, follows the northward route, and even the first transatlantic airship flight ever took a similar route. It could afford to use the quicker route instead of the more cloud-free route because the airship is designed to start with a fully charged battery. As the solar power system is capable of generating enough energy to recharge the batteries mid-flight, once they are full the airship can fly less ideal light scenarios without much problems. This can also be seen in the battery capacity plot in Figure 21. While the first few hours are actually draining the battery, the airship is able to charge it back up to full capacity on the same day before the night starts.
The second part of the route that experiences less than ideal light conditions is the part where the airship is flying through very high clouds at around 16 hours into the flight. When looking at the actual radiation that reaches the ship during that time, plotted in the left image of Figure 22, one can see that it is actually nighttime during that part of the flight. Therefore, the airship is not able to generate any solar power during that time anyway.
The third part of the route that experiences less-than-ideal light conditions is when the airship passes through a cloud layer at the very end of the flight. As the battery is relatively full and not much more flight time is expected, the airship can simply switch to battery power and traverse the last cloud layer.

7.4 Upward difussive radiation during the flight

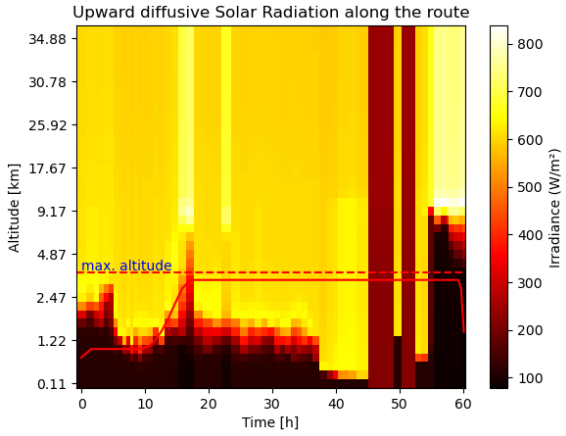

Figure 24: Upward diffusive radiation on a route from New York to London in the winter with a max height of $3500m$. For all the radiation simulations the sza has been set to 0.

Figure 24 shows the diffuse radiation values reaching the underside of the airship. As previously discussed in Section 9, the upward diffuse radiation can account for a significant portion of the total radiation that reaches the airship. In this case, this effect is even more pronounced as the airship is flying over clouds for

most of the flight. At nearly every node, the upward diffuse radiation values are high enough to warrant additional solar panels on the underside of the airship. While the direct and downward diffuse radiation values combined are often in the range of 1000 to 1200 W/m^2, the upward diffuse radiation values are often in the range of 600 to 700 W/m^2. This means that the airship could receive a significant amount of solar radiation from the underside. However, the addition of numerous extra solar cells could significantly increase the airship's weight. Therefore, evaluating whether the increase in solar power generation offsets the added weight is essential.

7.5 Circumventing clouds

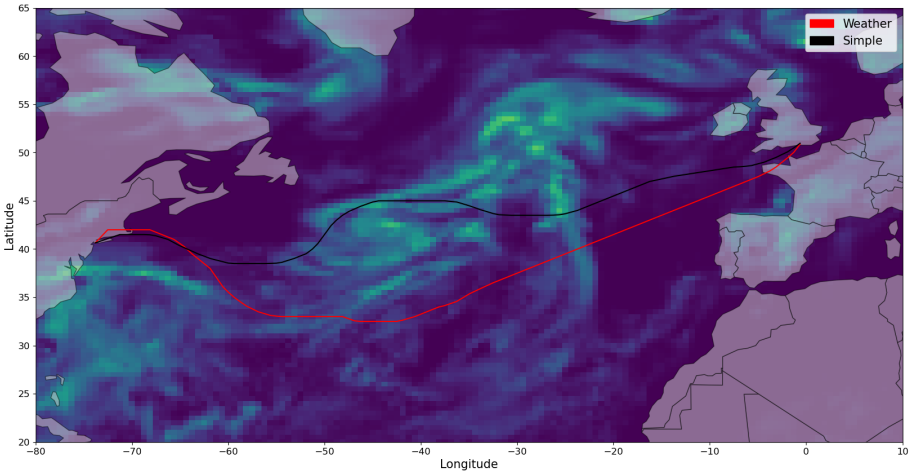

Figure 25: Route from New York to London in the winter with a max height of $3500m$.

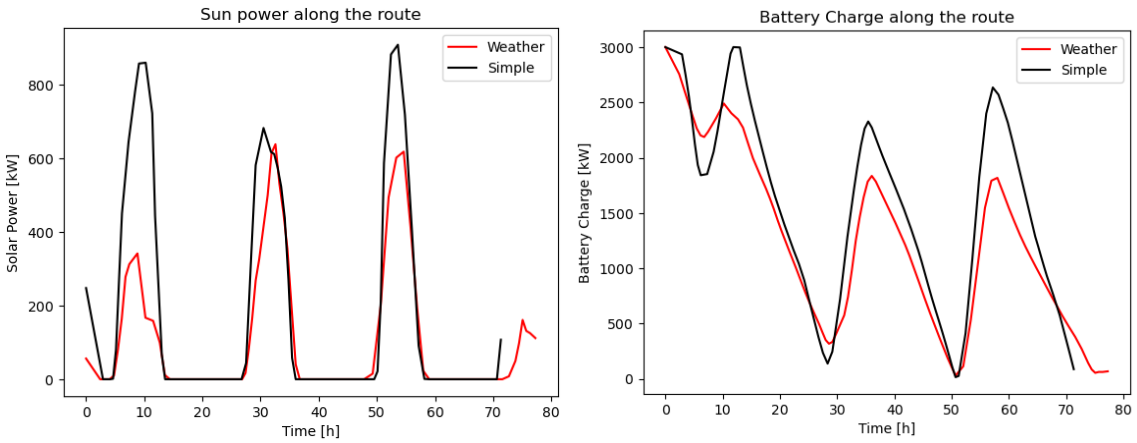

Figure 26: The solar power generated by the airship for a route from New York to London in the winter with a max height of $3500m$. The Simple version uses the original radiation approximation to calculate the solar power and the Weather version uses the new LibRadtran model.

In the previous example, the airship kept flying through the clouds even if it could have avoided them. This resulted in the same route as calculated by the older, much simpler radiation model. The reason for this is probably that the battery capacity was high enough to allow for short periods of no solar power generation. In order to see if the new radiation model can actually help the airship to avoid clouds, a different calculation has to be done that brings the battery's capabilities to its limits.
When producing significantly less solar power, the airship will have to follow a more optimal route. Instead of changing the airship's parameters to have a lower battery capacity or less solar area, one can instead change the incoming radiation by letting the airship fly in the winter. In Figure 25, one can see the resulting

routes for a flight from New York to London in the winter. Here, the new route is significantly different from the old one. It circumvents a majority of the clouds in order to be still able to generate enough solar power to keep the airship flying. Figure 26 shows that the solar power generated by the airship for the new route is significantly lower during the first day of flight and then roughly matches the solar power generated by the old route for the following days. On the first day, the battery capacity is drained to keep flying through the clouds before it is recharged to around 60% during the second day. It is important to remember that the "Simple" routes solar power data was calculated using the simple irradiance approximation that did not consider weather conditions. Therefore, the overall solar power generated by the airship is higher than using the new LibRadtran model.

7.6 Performance problems for low irradiance routes

Route	Simple	Weather
LO-NY Summer	4m41.005s	33m55.742s
LO-NY Winter	1m43.952s	1211m42.397s
NY-LO Summer	1m13.893s	47m40.346s
NY-LO Winter	1m25.182s	574m33.230s

Table 2: The runtime comparison between the simple solar power calculation and the LibRadtran model with weather data for different routes and seasons. In this example the weather data (wind, cloud coverage) is the same for all simulations. Run on an Intel Xeon Gold 6134 ("Skylake"), 8 cores @ 3.2GHz.

The new, more complex radiation model has a significant impact on the runtime of the airship pathfinder software. While the simple solar power calculation only takes a few minutes to calculate the solar power generated by the airship for a given route, the new LibRadtran model with weather data can take considerably longer. For summer routes, runtimes fall within the range of 30 to 50 minutes, which is still acceptable. However, calculating winter routes is significantly more time-consuming. For instance, the winter route from London to New York can take over 20 hours to compute. Such a runtime is no longer in the range of being acceptable for home computer use. All reported runtime values have been generated on the university's high-performance computing cluster. Unfortunately, as the current state of parallelization of the airship pathfinder software is not very good, the software cannot make full use of the available computing power and is not able to make use of more than one node. Additionally, the Intel CPU used for this software is not optimal, as it has a low core count and relatively low clock speed. Consequently, most modern home computers could likely achieve much better runtimes than those reported here.
This, however, does not solve the problem of the ridiculously long runtime for the winter routes. The problem with these routes lies in the fact that during the winter, the solar power generated by the airship is significantly lower than during the summer. As a result, the Pathfinder software must evaluate multiple routes to identify a suitable one. This leads to a substantial increase in radiation checks, resulting in more cache entries.

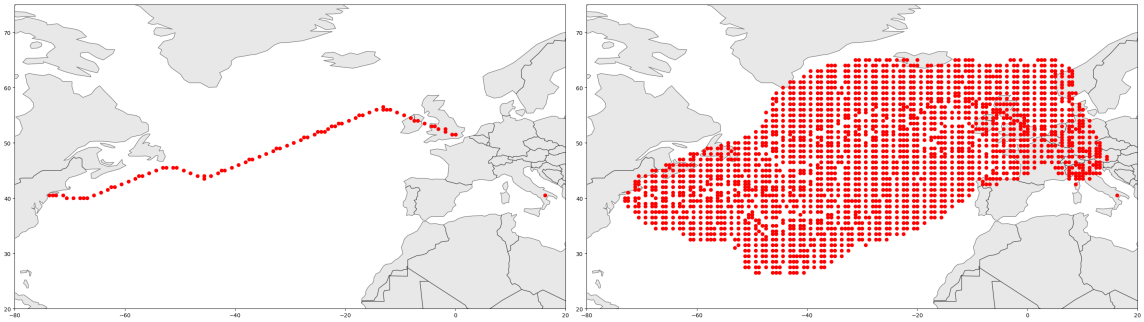

Figure 27: Radiation cache accesses for a route from London to New York in the winter with a max height of $3500m$.

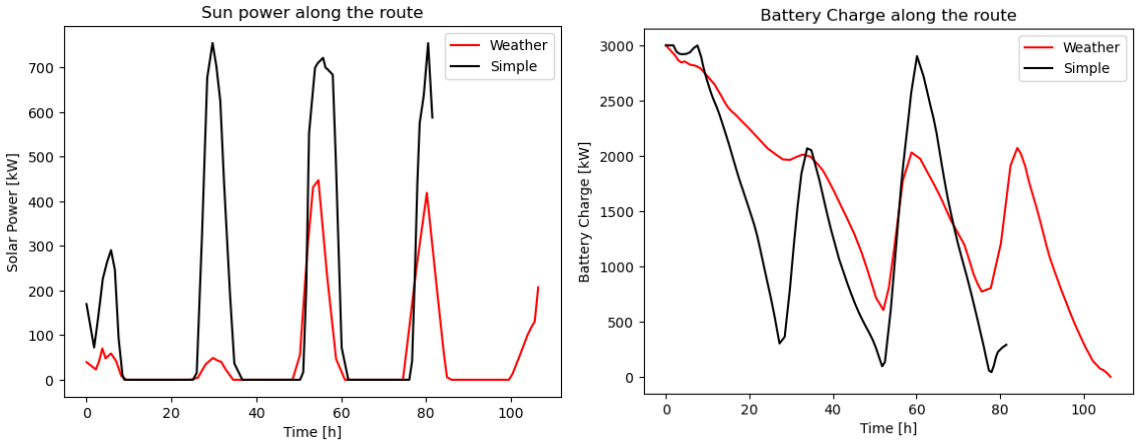

Figure 28: Solar power on the airship for a route from London to New York in the winter with a max height of $3500m$. Simple is the old and weather the new model with weather data.

Figure 27 shows the cache access patterns for the summer and winter routes from London to New York. While the summer route only has a few cache accesses along the final route, the winter route nearly checks all possible nodes between London and New York. This contributes to a significant increase in the software's runtime. It has to do this because the airship receives so little solar power that it has to fly a very specific route to be able to generate enough power to keep flying. This can be seen in the solar power plot in Figure 28. The plot shows the solar power generated by the airship for the winter route from London to New York. This can be compared to Figure 21 showing the solar power generated by the airship for the summer route. In the summer, the airship generates nearly identical solar power values for both the simple and weather models. However, during winter, the weather model produces considerably lower solar power values, leading to numerous irradiance checks for various potential routes.
Reducing this runtime on a home computer would require a rework of the Pathfinder software to make better decisions on which nodes to check. When using a high-performance computing cluster, however, the runtime could now be significantly reduced by calculating the field of radiation values for the space between London and New York once in the beginning and then using this data for the routes. If data is needed from outside this region, the software could then calculate this data on the fly. Calculating the radiation values for a whole grid of points would allow easy scaling of the software to more nodes. And as the cache currently calculates the radiation values for all times of the day anyway, this would allow routes to take slower or longer paths and still have a precalculated radiation value for each position, leading to a significant reduction in runtime. Alternatively, instead of calculating radiation values for all timesteps, the software could focus on computing values only for the current and surrounding timesteps. This approach could save considerable time on radiation calculations, although it would complicate cache access slightly and be more challenging to parallelize effectively.

8 Solar panel efficiency per wavelength

The previously described calculations for solar power use the entire light wavelength spectrum as radiation values to calculate the power produced by the solar panels. In reality, solar panels are more efficient at some wavelengths than others. The LibRadtran model is able to calculate the radiation values for various wavelengths, as seen in the right plot of Figure 17. In this plot, one can also see that the radiation values shift to longer wavelengths after passing through a cloud, as is evident by looking at the diffusive radiation values. The diffuse downward radiation values, in particular, show a significant shift to longer wavelengths. As the airship does not have the altitude capabilities to fly above all cloud layers, including this effect in the solar power calculations may be beneficial. It is possible to simulate the radiation for different wavelengths using LibRadtran and then multiply them with their respective efficiency values for the given solar panel configuration. The efficiency values for the different wavelengths for a given solar cell can be found in the external quantum efficiency (EQE) curve. However, unlike a typical efficiency value, the EQE values actually describe the number of electrons produced by the solar cell per photon. This means that, since the energy of a photon can exceed the energy required to produce an electron, the efficiency values can be significantly higher than 1. It is important to understand that the efficiency of a solar cell is not equal to the efficiency of a solar panel. This means that without also having the efficiency values for the solar panel configuration, it is not possible to calculate the power produced by the solar panels. Due to the lack of information regarding the solar panel configuration, the wavelength-dependent efficiency values are not currently utilized in the airship simulation.
Although the total solar output of the solar panels cannot be determined, the output of the solar cells can still be calculated in this manner. This allows for comparisons of solar energy potential under different weather conditions.

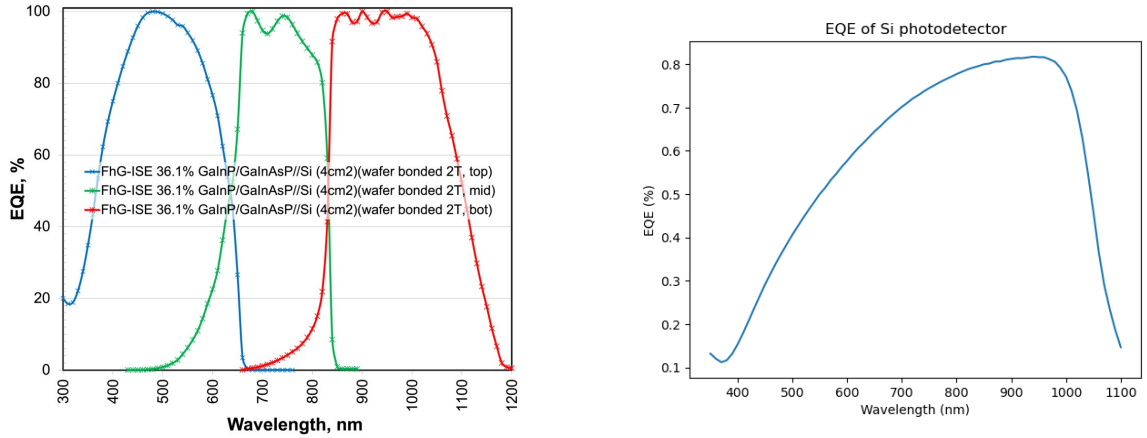

Figure 29: Left: The EQE curves for new 2-terminal triple-junction cells. Graphic by [9]. Right: The EQE curve for a FDS100-CAL SI photodetector.

Figure 29 presents the EQE curves for different solar cells. It's important to note that the right plot displays the EQE curve for an SI photodetector, not a solar cell. This data should, therefore, only be used as a basic reference point for the efficiency at different wavelengths. Using the provided EQE curves, one can see that different types of solar cells can have wildly different efficiency values at different wavelengths. Therefore, once the solar panel configuration is known, it could be an integral part of the airship route planning to also include the predicted solar power output of the solar panels for different weather conditions and their resulting shift in the radiation spectrum. Inversely, the solar panel configuration could also be optimized based on the expected weather conditions.

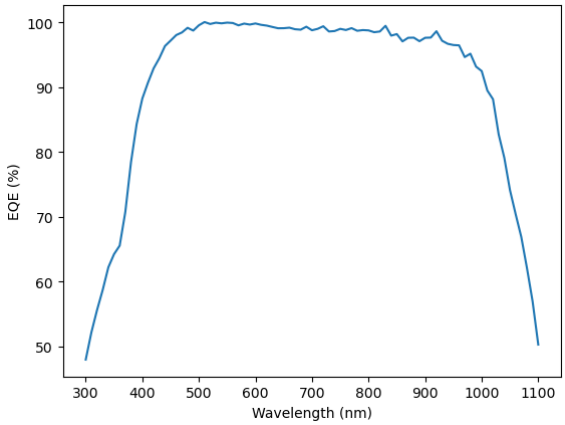

Figure 30: Measured EQE curve for a crystalline silicon solar cell.

Approximating the solar cell efficiency

As the solar panel configuration is unknown, the actual efficiency has to be approximated in order to calculate the power output of the solar panels. Manufacturers of solar panels often provide the efficiency values for the solar panels for a given laboratory condition. While this helps to compare different solar panels, it does not help to calculate the power output of the solar panels for different weather conditions. Some cloud types can have a significant effect on the radiation spectrum that reaches the solar panels of the airship. Therefore, in order to further increase precision in the solar power calculations, the efficiency values for the solar panels have to be known for each wavelength. Sadly, this information is often not readily available and has to be measured in a laboratory for each solar panel configuration. As this is not feasible for the purpose of this thesis, the efficiency values for the solar panels have to be approximated. The efficiency of a solar module is dependent on its structure. Therefore, it is not possible to get a precise value for its efficiency. However, the efficiency of a solar cell is only dependent on the material it is made of and can be calculated using the EQE curve. So, instead of trying to calculate the efficiency of the solar panels, the efficiency of the solar cells can be calculated and used to compare the solar power output of the airship for different weather conditions. We then treat the solar panel as a perfect solar cell and use the calculated efficiency values to get a rough estimate of the solar power output. Once a solar panel configuration is known in the future, real values can be used instead.
Calculating the efficiency of a solar cell per wavelength can be done by using its EQE curve. This curve is often not readily available and has to be measured in a laboratory. Since EQE values only indicate the number of electrons produced by the solar cell per photon, they must first be converted into efficiency values. To address this, I propose the following method:

To calculate the resulting power output of a solar cell for a given frequency using the radtran output one has to first convert it to the number of photons per second per area equivalent.

$$N_{\text{photon}} = \frac{W\lambda}{h \cdot c}$$

where W is the LibRadtran output, λ is the wavelength and h is the Planck constant. Note that the actual LibRadtran output is the irradiance in $\frac{W}{m^2} = \frac{J}{s \cdot m^2}$ and not J. Therefore, this calculation determines the number of photons per second per unit area, but for simplicity, it is denoted as N_{photon}. The number of photons per second per area can then be converted to the number of electrons per second per area using the EQE curve.

$$N_{\text{electron}} = N_{\text{photon}} \cdot \text{EQE}_{\lambda}$$

Multiplying this value with an electron's elementary charge gives the solar cell's short-circuit current density.

$$J_{SC} = N_{\text{electron}} \cdot e$$

The short-circuit current I_{SC} represents the maximum current that could theoretically be generated if there were no resistance in the system, effectively resulting in a short circuit. This current can be calculated by

multiplying the short-circuit current density by the area the solar radiation acts upon. This area is defined by LibRadtran as $A = 1m^2$:

$$I_{SC} = J_{SC} \cdot A \rightarrow I_{SC} \hat{=} J_{SC}$$

For simplicity, we will assume that the illuminated current $I_L \approx I_{SC}$. The open-circuit voltage V_{OC} is the maximum voltage that can be obtained from the solar cell when no current is flowing.

$$V_{OC} = \frac{n \cdot k \cdot T}{q} \ln\left(\frac{I_L}{I_0} + 1\right)$$

where n is the ideality factor, k is the Boltzmann constant, T is the temperature, q is the elementary charge and I_0 is the saturation current. For our model, we will consider the solar cell to be ideal, setting the ideality factor to $n = 1$. The temperature T will use the temperature at the current position and altitude provided by the MERRA-2 dataset. The saturation current I_0 is a material parameter that is an indicator of the quality of the used silicon in the solar cell. For the purpose of this calculation, we will use a value of $I_0 = 10^{-10}A$.
The Fill Factor FF represents the area of the largest rectangle that can be drawn in the I-V curve of the solar cell.
It can be calculated by using Lamberts functions [10]:

$$FF = \frac{v_{OC} - \ln(v_{OC} + 0.72)}{v_{OC} + 1}$$

with v_{OC} being the normalized open circuit voltage.

$$v_{OC} = \frac{q}{n \cdot k \cdot T} V_{OC} = \ln\left(\frac{I_L}{I_0} + 1\right)$$

Finally, the efficiency η of the solar cell can be calculated by:

$$\eta = \frac{V_{OC} \cdot I_{SC} \cdot FF}{P_{in}}$$

with P_{in} being the solar cell's input power and the output of the LibRadtran model.

9 Diffuse radiation

In order to calculate the correct power output of the solar panels on the airship, it is necessary to know how much light is actually reaching these solar panels. The total irradiance reaching the solar panels can be calculated using the LibRadtran model. This model separates irradiance into three types: direct irradiance, diffuse upward irradiance, and diffuse downward irradiance. All radiation that reaches a given position without being reflected, scattered, or changed in any way is called direct irradiance. As this radiation is emitted from the sun and reaches the airship unobstructed, its direction is known, and it can easily be applied to the complete sun-facing solar area of the airship. The diffuse irradiance is the radiation that reaches a given position after being influenced by the atmosphere, clouds, or by the ground. This radiation is no longer coming from the direction of the sun but can come from any direction. Here, LibRadtran differentiates between the upward and downward diffuse irradiance, defining them as the integral of the radiance over their respective hemispheres. Specifically, diffuse downward irradiance is composed of all radiance values coming from the sky down to the horizon, resulting in the irradiance that strikes a plane oriented directly upward. Unfortunately, most of the solar panels on an airship will not be facing straight up but will be tilted by varying degrees. This means that the diffuse irradiance values calculated by the LibRadtran model are not useful for the airship simulation. Therefore, it is necessary to compute these values for a range of different tilt angles.

9.1 Irradiance vs radiance

In order to calculate the diffuse irradiance, LibRadtran provides a way of calculating the radiance values for given angles μ and φ. To make use of these radiance values, it is helpful to understand their difference from the required irradiances. The irradiance is the power per unit area that reaches a surface measured in W/m^2. The radiance is the power per unit area per unit solid angle emitted or reflected from a surface measured in $W/(sr\ m^2)$. A steradian is the unit of solid angle and is defined as the solid angle that subtends a surface area of $1m^2$ at a distance of $1m$. Integration is needed to convert the radiance values to irradiance values.

9.2 Calculation of the diffuse radiation

The uvspec program provided by the LibRadtran software suite uses the LibRadtran library to compute values for the direct radiation as well as two types of diffuse radiation in its default setup. These two diffuse radiation types are the upward and downward diffuse radiation. They are useful for calculating the radiation that reaches a solar panel that is facing upwards or downwards. However, most solar panels on an airship are mounted on the hull and, therefore, tilted at various angles rather than facing directly up or down. As a result, the diffuse radiation values calculated by the uvspec program are not applicable to these panels. To effectively assess the radiation on these tilted panels, it is necessary to compute the diffuse radiances for different angles. Unfortunately, the uvspec program does not provide these values, and there is no configuration to obtain them. It does, however, provide the values for the radiance at different angles μ, φ.

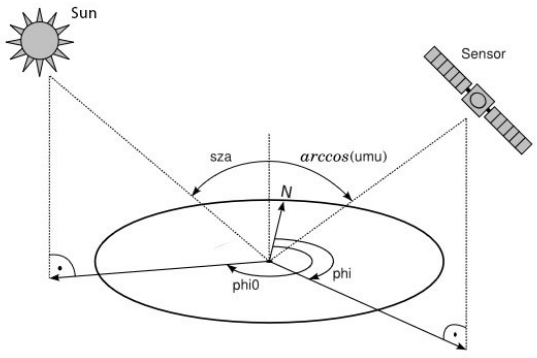

Figure 31: Parameters that are used to describe the radiance calculation of the uvspec program. Graphic by [11]

The radiance data can be calculated by providing the uvspec program with the angle of the sun φ_0 and the

relative angle of the sensor φ. A value of $\varphi_0 = \varphi$ means that the sensor faces the sun directly, and a value of $\varphi_0 - \varphi = 180\,\text{deg}$ means that the sensor is facing away from the sun. The solar zenith angle sza represents the sun's altitude angle, while the sensor's angle is represented by the viewing zenith angle θ. Instead of the angle θ, the uvspec program requires the input value $\mu = \cos(\theta)$. Given the two angles μ and φ, the uvspec program will return the radiance for the given direction in milliwatts per steradian per square meter or $W/(sr\ m^2)$. If multiple angles are provided, the uvspec program will return the radiance values for each permutation of them.

```
lambda edir edn eup uavgdir uavgdn uavgup
                      phi(0)              ... phi(m)
umu(0) u0u(umu(0)) uu(umu(0),phi(0)) ... uu(umu(0),phi(m))
umu(1) u0u(umu(1)) uu(umu(1),phi(0)) ... uu(umu(1),phi(m))
. . . .
. . . .
umu(n) u0u(umu(n)) uu(umu(n),phi(0)) ... uu(umu(n),phi(m))
```

Figure 32: Output structure of the uvspec program for multiple radiance angles.[7]

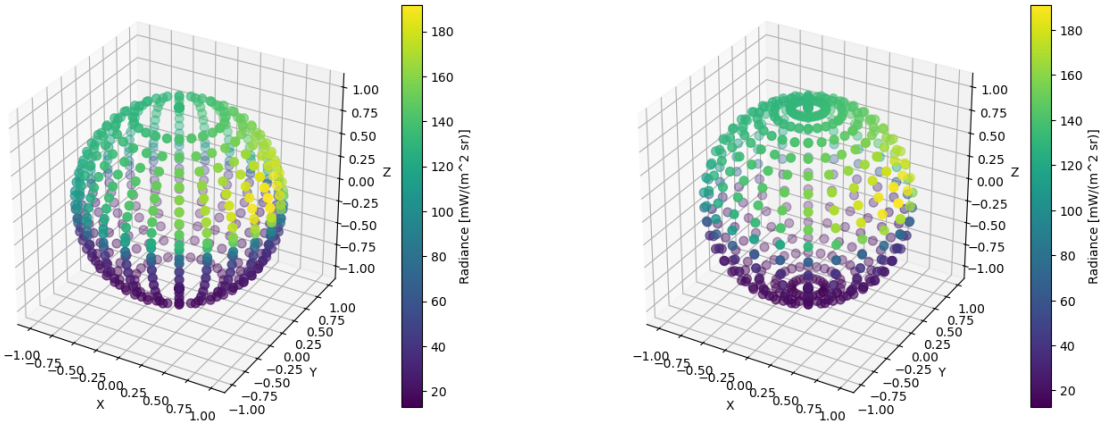

Figure 33: Generated using the cloud data from Figure 18 with a sza of $40\,\text{deg}$. Both images show the uvspec radiance output for a given set of $\varphi \in [0, 360], \mu \in [-1, 1]/0$. Left: The uvspec radiance output for φ stepsize of $15\,\text{deg}$ and a μ stepsize of 0.1 with an angle 0.05 instead of 0. Right: The uvspec radiance output for φ stepsize of $15\,\text{deg}$ and 20 equally distributed steps of $\theta = \arccos(\mu)$.

Using the weather data shown in Figure 18 and an sza of $40\,\text{deg}$ the uvspec program was run for the angles $\mu = \cos(\theta) = [0, 1]$ and $\varphi = [0, 360]\,\text{deg}$. The result of this calculation can be seen in Figure 33. In the left image, one can see that the resolution for the μ angles is very low at the poles. This happens because this image was generated for a μ stepsize of 0.1 in the range $[0, 1]$. But with $\mu = \cos(\theta)$, this actually leads to a very irregular step distribution for the θ angles. So, in order to fix this, one can simply use an arbitrary stepsize for the θ angles and convert them to μ for the uvspec program. This can be seen in the right image of Figure 33. The resulting radiance values are now more evenly distributed over the whole sphere. Because uvspec generates a radiance entry for every permutation of the given angles, the resulting data field, if plotted on top of a sphere, features high resolution at the poles and lower resolution at the equator. Similar to the longitude lines on a world map, the resulting data points are closer together at the poles and further apart at the equator. This means that by trying to increase the precision at the equator, there will be a significant increase in the amount of data points generated even at the poles where the resolution is already extremely high. While this is currently not a big problem as this number of data points are still manageable, it could become a problem if the resolution is increased even further. Also, once integrated into the airship simulation, the radiance values will have to be interpolated for all the angles of every single solar panel for every radiation calculation. This results in a significant increase in the amount of data that has to be processed, which could benefit from a more efficient data point structure.

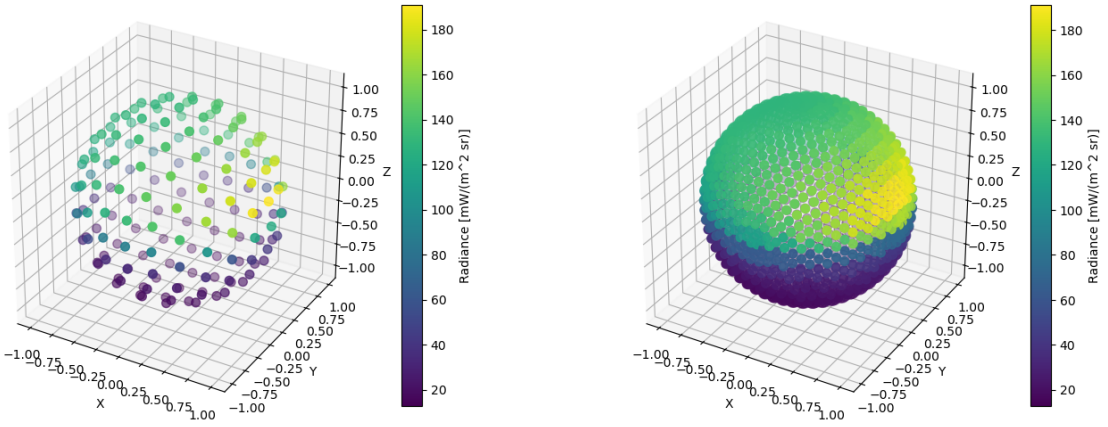

Figure 34: Lebedev grid representation of the data in Figure 33 of order 21 and 65 with 170 and 1454 points respectively.

By using the Lebedev grid, one can generate a more evenly distributed set of data points for the radiance values. Two examples of this can be seen in Figure 34. Such a point cloud allows for the increase or decrease of the number of points without creating huge areas of low precision or extremely high precision. Representing the data like this also allows for easy integration over parts of the sphere, which is needed for the diffuse radiation calculations. When integrating over the entire sphere, we expect to obtain the same radiance value as the output from the uvspec program for combined upward and downward diffuse radiation. Similarly, integrating over just the top half of the sphere should yield the same radiance value as that for upward diffuse radiation and vice versa for downward diffuse radiation. This can be used to verify the correctness of the data and the integration process.

9.2.1 Interpolation

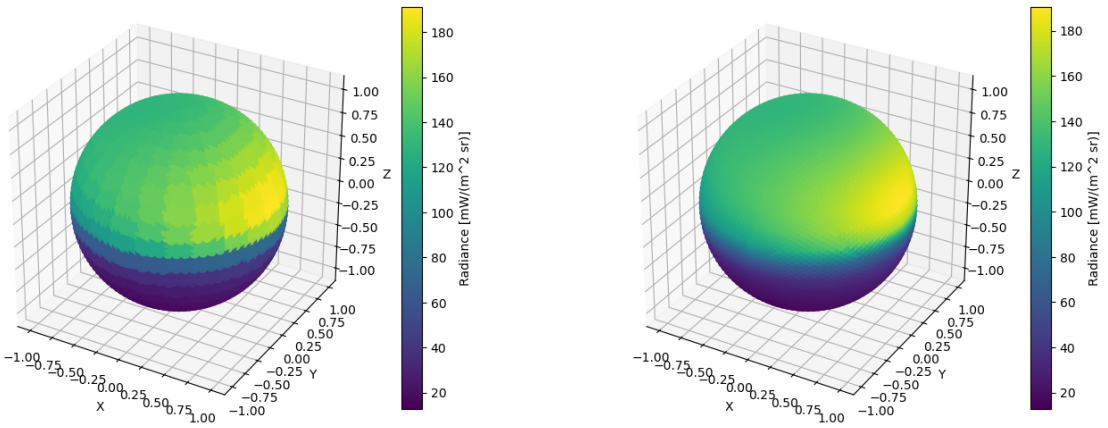

Figure 35: Lebedev grid of order 131. Left with nearest neighbor interpolation and right with linear interpolation.

The LibRadtran library is able to calculate the radiance values for a given set of angles μ, φ. However, using the uvspec program, obtaining the values for those angles is not possible. Instead, one has to provide the different angles μ and φ separately using the uvspec input file. The program will then calculate the radiance values for all the permutations of the given angles. This results in a set of radiance values that are not evenly distributed over the whole sphere. In order to still make use of the advantages of the Lebedev grid, the radiance values have to be interpolated from the uvpsec output. Two different interpolation methods have been tested for this purpose. The first one is the nearest neighbor interpolation. It provides an easy way to get the radiance values for a given angle by simply finding the closest data point and using its value. As the radiance values are provided in a grid-like structure, see Table 3, finding the closest data point is a

simple and runtime-efficient task. While this approach is the simplest and fastest interpolation method, it does not provide the best results. The resulting radiance values appear very blocky and do not represent the actual change in radiance very well. The size of these blocks is also very dependent on the resolution of the underlying data points. As such, the precision is very high at the poles and very low at the equator. An example of this can be seen in the left image of Figure 35. The second method tested is linear interpolation, which offers a more accurate representation of the radiance values. This method uses surrounding data points to calculate the radiance value for a given angle. The resulting values are much smoother and represent the actual change in radiance much better.

	φ_0	φ_1	$\cdots$	φ_m
μ_0	$uu(\mu_0, \varphi_0)$	$uu(\mu_0, \varphi_1)$	$\cdots$	$uu(\mu_0, \varphi_m)$
μ_1	$uu(\mu_1, \varphi_0)$	$uu(\mu_1, \varphi_1)$	$\cdots$	$uu(\mu_1, \varphi_m)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
μ_n	$uu(\mu_n, \varphi_0)$	$uu(\mu_n, \varphi_1)$	$\cdots$	$uu(\mu_n, \varphi_m)$

Table 3: Output structure of the uvspec program from Figure 32 rewritten as a table to show a more ordered representation of the calculated radiance values.

It should be noted that while Table 3, showing the radiance values for the angles μ, φ in a grid-like structure, may suggest that the radiance values are distributed evenly over the whole sphere, this is not the case. For an example of this, see Figure 33.

9.2.2 Lebedev Integration

In order to calculate the irradiance from the radiance values, the radiance values have to be integrated over the whole or parts of the sphere.

$$E = \int L(\theta, \varphi) d\theta d\varphi$$

Since there is no analytical function to integrate, but rather a set of data points, the integration must be performed numerically. By using the radiance data from Figure 34, the values can be integrated using the Lebedev quadrature. This method allows us to approximate the surface integral as a weighted sum of the data points.

$$E \approx 4\pi \sum_{i=1}^{N} w_i L(\theta_i, \varphi_i)$$

Here, w_i represents the weights of the Lebedev quadrature, while $L(\theta_i, \varphi_i)$ denotes the radiance values at the given data points. The Lebedev quadrature requires only one summation to compute the surface integral, making it relatively efficient.

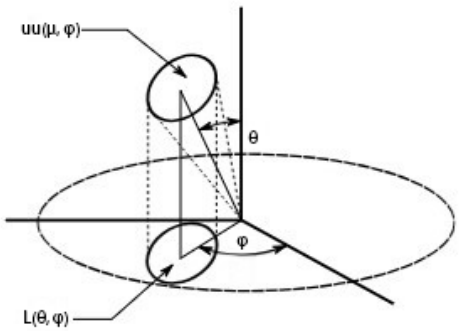

Figure 36: The radiance parameters. Graphic by [11]

The uvspec program provides the radiance values for the given angles θ, φ as $uu(\mu, \varphi)$, where $\mu = \cos(\theta)$. The output file structure can be seen in Figure 32. Figure 36 shows the parameters that are used to describe

the radiance calculation of the uvspec program in a more graphical setting. There, it can also be seen that $L(\theta, \varphi)$ is the projection of $uu(\mu, \varphi)$ onto the unit circle, with its normal vector corresponding to the view direction. This means that the view direction of a solar panel is the same as the normal vector of this unit circle. As the size of the projection $L(\theta, \varphi)$ is solely dependent on the angle θ and not on the angle φ, its conversion can easily be written as $L(\theta, \varphi) = uu(\mu, \varphi)|\cos(\theta)|$. This leads to the following formulation for the numerical integration of radiance values over the entire unit sphere:

$$E \approx 4\pi \sum_{i=1}^{N} w_i uu(\mu_i, \varphi_i)|\cos(\theta_i)|$$

and with $\mu = \cos(\theta)$:

$$E \approx 4\pi \sum_{i=1}^{N} w_i uu(\mu_i, \varphi_i)|\mu_i|$$

In order to solve this equation, one still needs the weights w_i of the Lebedev quadrature. As the weight values for a given set of points are always the same, a precomputed set of weights is used. The actual values of the weights have been visualized in Figure 37.

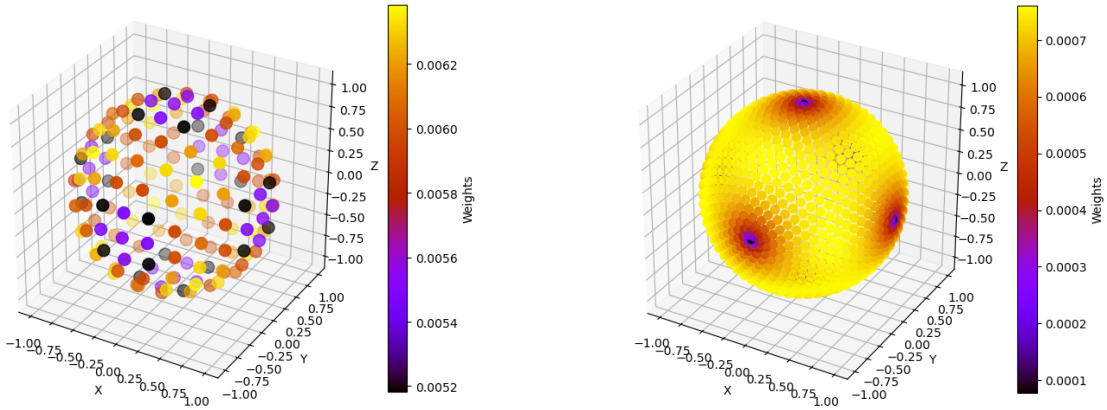

Figure 37: Both images show the values of the Lebedev weights for a given order. The left figure features an order of 21 with 170 points and the right figure an order of 65 with 1454 points.

To integrate only specific parts of the sphere, we can set the radiance values of the regions that are not of interest to zero. This approach may introduce small errors in the calculations. When the points are located at a greater distance from each other, the algorithm might either overestimate or underestimate the results. However, with a sufficient number of data points, this method should yield acceptable results. Using this technique, we can calculate the irradiance values for a hemisphere oriented in the direction of μ_0, φ_0 by:

$$E_{\text{direction}}(\mu_0, \varphi_0) \approx 4\pi \sum_{i=1}^{N} w_i \begin{cases} uu(\mu_i, \varphi_i)|\mu_i| & \angle(\theta_i, \varphi_i)(\theta_0, \varphi_0) \leq \frac{\pi}{2} \\ 0 & \text{else} \end{cases}$$

9.2.3 Precision of the Lebedev quadrature

Interpolation	Result					
Nearest Neighbour	Direction	Reference	Order 9	Order 21	Order 65	Order 131
	Downward	65.1538	71.5529	73.7060	65.1350	65.5782
	Upward	444.6903	426.6887	440.9565	444.1421	444.5354
	Total	509.8441	484.1415	506.4533	509.1619	510.0992
Linear	Direction	Reference	Order 9	Order 21	Order 65	Order 131
	Downward	65.1538	69.3827	72.4010	66.0833	66.0552
	Upward	444.6903	425.5008	442.1146	443.8327	443.9656
	Total	509.8441	480.7833	506.3063	509.8008	510.0064

Table 4: The results of the Lebedev quadrature for different orders and interpolation methods. With reference being the values calculated by the uvspec program. Generated using the cloud data from Figure 18 with a sza of 40.

As the Lebedev quadrature is a numerical integration method, the precision of the results depends on the number of points used. In Table 4, the results of the Lebedev quadrature for different orders and interpolation methods can be seen. The reference values are the values calculated by the uvspec program. The results indicate that the precision of the Lebedev quadrature increases with the order of the quadrature. While the order 9 and 21 methods feature some relatively high errors of roughly 10% for the diffuse downward radiation, the order 65 and 131 methods feature errors of less than 1%. The increase in error for the 21-order method is most likely due to the fact that in order to calculate the integral for only one-half of the sphere, the radiance values for the other half of the sphere have been set to 0. If the position of the points results in an unfavorable distribution, it could lead to more points being used for the integration of the downward radiation than the upward radiation or vice versa. Therefore, this creates a higher error in the result. Increasing the number of grid points mitigates these issues, as demonstrated by the improvements seen with the 65-point and 131-point methods.
In order to calculate the diffusive upward and downward radiation values for the solar panels on the airship over their hemispheres, previous general formulas can be simplified. Because the values for μ_0, φ_0 are now $1, 0$ and $-1, 0$, the angle φ no longer needs to be checked, and the selection condition can therefore be trivialized. The formulas can be simplified to:

$$E_{\text{downward}} \approx 4\pi \sum_{i=1}^{N} w_i \begin{cases} uu(\mu_i, \varphi_i)|\mu_i| & \mu_i < 0 \\ 0 & \text{else} \end{cases}$$

$$E_{\text{upward}} \approx 4\pi \sum_{i=1}^{N} w_i \begin{cases} uu(\mu_i, \varphi_i)|\mu_i| & \mu_i > 0 \\ 0 & \text{else} \end{cases}$$

$$E_{\text{total}} \approx E_{\text{downward}} + E_{\text{upward}}$$

In the displayed equations, the zero μ values are not included in the integration. This was done to be more in line with the LibRadtran model because the model does, per documentation, not support an angle of $\mu = 0$. As we multiply with $|\mu|$, the zero values would not have contributed to the result anyway.

9.2.4 Directional diffusive irradiance per frequency

The LibRadtran model can not only calculate the total radiation reaching a specific location but can also provide this information on a per-wavelength basis. In that case, the previously described formulas can be used to calculate the irradiance values for the solar panels on the airship at different wavelengths. This can then be used to calculate the power output of the solar panels even more precisely by using the efficiency curves of said panels. Sadly, when doing both the efficiency calculation per wavelength and the directional diffusive irradiance calculation, the amount of data that has to be processed increases significantly. As a result, not only does the size of the data from each calculation grow substantially, but the runtime for these calculations and for the uvspec program itself also increases.
For example, the LibRadtran calculation alone using the cloud data from Figure 18 with a sza of 40 with a φ stepsize of $15\,\text{deg}$ and 20 equally distributed steps of $\theta = \arccos(\mu)$ for the wavelength range of

$280 - 1300nm$ took around $2.8\,\mathrm{sec}$. Now calculate the same result but for every wavelength in the range of $280 - 1300nm$ separately, and the runtime increases to around $75\,\mathrm{sec}$. Although this runtime is still manageable, it represents a significant increase by a factor of about 27. With the radiation calculation already being the most time-consuming part of the airship simulation, this increase is significant.
This is also not the only part of the radiation calculation that will increase its runtime. The airship software now has to do the same calculation it previously did, but now for every wavelength. The amount of output data generated by the radtran program increases from 32 lines to nearly 23000. While the resulting calculation of solar power output may be more precise, the overall runtime of the airship simulation rises substantially. Achieving such accuracy in solar power calculations is of limited use if errors introduced elsewhere in the airship simulation are significantly larger. For example, the fact that the grid resolution of the simulation is in the order of kilometers may introduce a much higher error than the solar power calculation. Although performing these calculations for every wavelength is possible and yields high accuracy, it comes with a significant cost in terms of program runtime. It should, therefore, be primarily used for a more intricate analysis of the solar power reaching the airship and to help find out the best configuration of solar panels for specific routes and regions. However, it is not suitable for the actual route-finding process.

10 Approximating the LibRadtran results using a neural network

The new radiation calculations using the LibRadtran library are significantly more accurate than the old calculations. However, they come at the cost of a much higher runtime, as seen in Table 2. While runtimes of around $25\,\mathrm{min}$ are acceptable for a single route calculation, certain routes now require over 20 hours of runtime to calculate. This represents a nearly 1000 fold increase in runtime compared to the older calculations. Currently, the user has the option of choosing the old, very fast, but less accurate calculations or the new, more accurate, but much slower calculations.
While deciding between speed or precision sounds like an acceptable tradeoff, the reality is that the simple approximation results in drastically different radiation values and, therefore, results in completely different routes. When one usually chooses between speed and precision, one would expect the results to be relatively similar but small differences in the details. As the approximation does not care for the actual weather data, the results naturally have to be different.
Therefore, it would be beneficial to have a model that can approximate the LibRadtran results with a much lower runtime. To address this, a neural network was trained on the LibRadtran results to predict the radiation values.
It should be noted that the idea of using a neural net to approximate the LibRadtran simulation resulted from a last discussion at the end of the thesis writing process and was purely intended as a small proof of concept. With roughly a week remaining to the submission, I was not able to finish writing the thesis and create complex neural networks at the same time. Therefore, the neural networks used in this thesis are relatively simple and feature no extensive hyperparameter optimization.

10.1 Generating the data

In order to train a neural network, it is first necessary to have a set of data that can be used for training. Since we aim to predict the outcomes of the LibRadtran simulation, we can create this dataset by running the simulation multiple times with various inputs. These inputs will also serve as the inputs for the neural network.

```
sza, height, day_of_year, surface_altitude, LWC_1, ..., LWC_42, IWC_1, ..., IWC_42
```

The inputs for the LibRadtran simulations include the solar zenith angle (sza), the altitude of the airship, the surface height, the day of the year, and the cloud data. The cloud data is characterized by the liquid water content and the ice water content of the clouds, with values provided for 42 different layers of the atmosphere. This setup results in a total of 88 input values for the neural network.
For each of these inputs, the LibRadtran simulation will be called and generate an output file with the same structure as already shown in Figure 32. Therefore, every output file will also feature the radiances that are needed for the interpolation discussed in Section 9.
The input values for each calculation are randomized. Specifically, the sza values are sampled from a uniform distribution ranging from $0\,\mathrm{deg}$ to $90\,\mathrm{deg}$, the day of the year is chosen between 1 and 365, the surface height is varied between 0 and 3 km, and the airship altitude is set between 0 and 5 km. The airship altitude was additionally set to always be higher or equal to the surface height. Every of these input vectors is appended a randomly selected cloud data file as a further input. These cloud files were all sourced from random positions on the same day from the MERRA-2 dataset. As one single MERRA-2 timestep already contains more than 300000 cloud entries there were no further augmentation techniques used on this data. The resulting dataset features more than 100000 entries, which has been split into training and validation set with a ratio of 80% to 20%.
Additionally, a second test dataset was created using the same randomization approach as the first dataset, but this time with cloud data from a different day.

10.2 Predicting the irradiances

Radtran provides three irradiance values for each simulation: direct, downward diffuse, and upward diffuse irradiances. The neural network was trained to predict these three irradiance values. As a network, a simple Multi-Layer Perceptron (MLP) was chosen as a network architecture. It was trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 128. After training the resulting network featured a mean squared error of 0.059 for the training set, 0.0786 for the validation set and 0.10 for the test set.

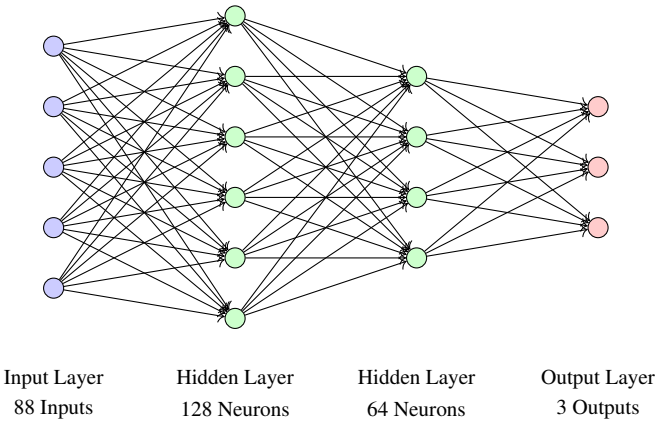

Figure 38: The MLP model used to predict the irradiances. It consists of only fully connected layers, each with a ReLU activation function.

Example results can be seen on the right side of Figure 39, with values from the LibRadtran simulation shown on the left side. The network was not trained, validated, or modified in other ways that are dependent on the shown values. After the model was determined that featured the best validation score it was then used to generate the data for this route.
Because the model was trained on maximum flight altitudes of 5km, the results for higher altitudes are not included. Looking at the results, one can see that the neural network is able to predict the values of the LibRadtran simulation quite well. It however seems to fail near the end of the route where conditions fluctuate from no clouds to complete cloud cover.
In the section with no clouds, the direct radiation predicted by the neural network significantly diminishes near the surface. This is probably a side effect of the training data as most of the training data featured clouds and therefore nearly always resulted in no direct sunlight at lower altitudes. This may also explain why the model tends to overestimate the downward diffusive light near the surface even if there is no cloud present. The upward diffuse irradiance exhibits a similar issue. In cloud-free conditions, the value returned by LibRadtran remains constant, as illustrated by the dark red bars on the last plot on the left. Notably, these bars are absent in the model's approximation.
Since all these bars have the same value and only appear when there is no cloud coverage, they should be easily identifiable and could be filled in manually. However, the model's failure to capture this suggests imbalances in the training data that may need to be addressed or supplemented with post- or preprocessing in conjunction with the neural network.
Overall, the average values produced by the model are slightly lower than those calculated by LibRadtran, but the overall shape of the results in the direct and downward diffuse plots is very similar.

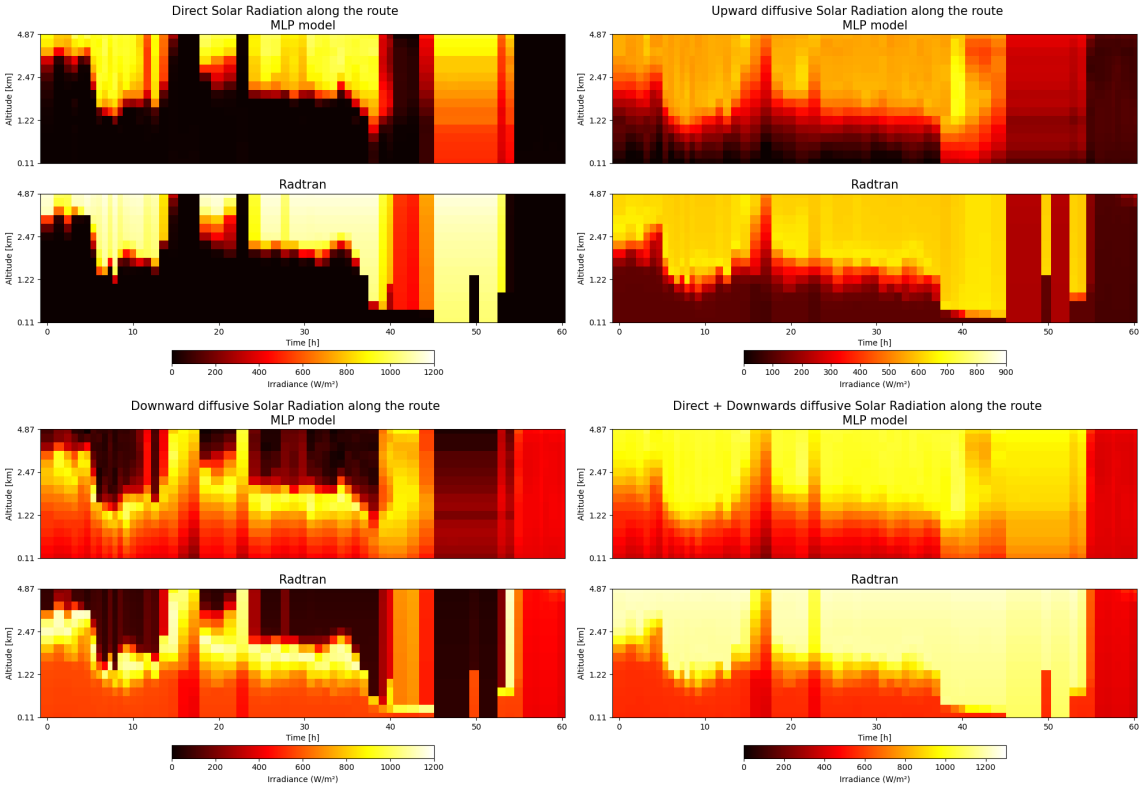

Figure 39: Plots showing the AI prediction and the LibRadtran results side by side. The top plot presents the AI prediction, while the plot below it shows the LibRadtran reference solution. There are four plots in total, arranged from top to bottom and left to right, illustrating the following: direct irradiance, downward diffuse irradiance, upward diffuse irradiance, and the sum of direct and downward diffuse irradiances. This is the same data as already shown in Figures 23 and 24. As the neural net was not trained on altitudes higher than 5km, so results for higher altitudes are not included.

10.3 Predicting the radiances

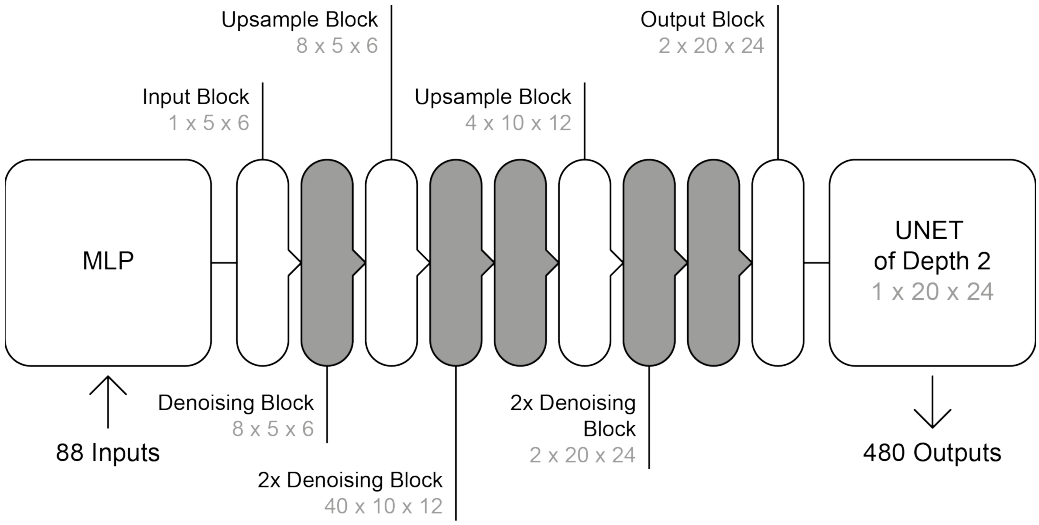

Figure 40: The AI model of the radiance prediction. The model features a Input-, Denoising-, Outputs-Blocks and an UNET based on the architecture based on the work of [12]. The MLP consists of hidden layers of size 128, 256, 256, 128, 30.

The section 9 used the radiance outputs of LibRadtran to calculate the irradiances for different angles. This was achieved by interpolating the radiance values using the Lebedev quadrature. The following neural net was trained to predict these radiance values. The input values for the neural network are still the same as those used in the previous network. Here, however, the output values are the radiance values for the angles μ, φ.

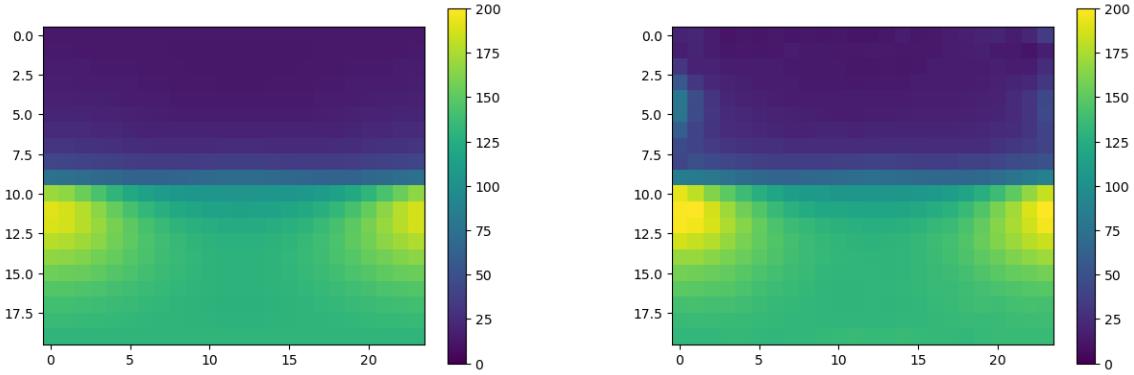

Figure 41: Visualization of the radiance values in $\frac{W}{sr\ m^2}$ that would reach the airship at different angles using the parameters from Figure 32. Image generated using φ on the x-axis and μ on the y-axis. Left the results calculated by the LibRadtran simulation and right the predicted values of the ai model.

The resulting field of radiance values can be represented as an image with the φ values on the x-axis and the μ values on the y-axis. An example is shown in Figure 41. On the left side, the results calculated by the LibRadtran simulation can be seen, and on the right side, the predicted values of the AI model. Much like with the previous example for the irradiance model, the data shown was neither part of the training nor the validation set. The same data was used as already used in a previous section and plotted as points on the surface of a sphere in Figure 33. Both figures represent the same data points. Apart from a few minor pixel values in the upper part of the image, the LibRadtran and the AI model values look very similar. This shows that the AI model may be able to predict the radiance values of the LibRadtran simulation with acceptable precision.

11 Summary

In this thesis, a more sophisticated model for the calculation of solar radiation reaching an airship was developed and integrated into an existing airship pathfinder software. Instead of relying on a simple approximation using a few equations, the new model utilizes the LibRadtran software for a more accurate simulation of the solar radiation reaching the airship. Using this model, it is possible to determine the irradiance for a specific position, altitude, and time of day. It even allows for effects like the distance between the sun and the earth to be taken into account. The new model was tested against the old one, showing more realistic results and better handling of different weather conditions. By using the MERRA-2 weather data, a dataset provided by NASA that contains the global weather data for a given day, in conjunction with the LibRadtran model, it is now possible for the airship pathfinder to take into account local weather phenomena during the calculations. This not only improves the accuracy of the expected solar power that can be generated by the airship for a given route but also allows for better route planning and optimization. As this new model requires a significantly higher runtime than the old model, the software was also optimized to use a caching algorithm to reduce the amount of radiation calculations that have to be done. By using this cache it was possible to reduce the number of needed LibRadtran simulations from 120 million to around 2160 depending on the selected route. While the resulting runtime is still significantly higher than the old model, it is now possible to calculate the solar power output for a given route in a reasonable amount of time. The integration of the new model into the airship pathfinder software was successful, and the software is now able to use more accurate solar power values for its route calculation.
In order to further improve the accuracy of the amount of radiation reaching the airship, the diffuse irradiance values were calculated for different angles. This was done by using the radiance values provided by the LibRadtran software and interpolating them using the Lebedev quadrature. The results show that the precision of such an approach is well within the required limits for the airship simulation.
It was also discussed how the new model can be used to calculate the solar power output of the airship for different wavelengths. With this approach, it is possible to calculate the power output of the solar panels even more precisely by using the EQE or efficiency curves of a given solar panel. Depending on the weather conditions and the time of day, some wavelengths may be more prevalent than others. This makes it possible to compare different solar panels and their efficiency curves to find the best solar panel for a given route. This might be useful for a more detailed analysis of the solar power output, but it also increases the runtime of the simulation significantly.
Additionally, a new method for calculating the solar area of the airship was implemented. This method uses the 3D model of the airship to calculate the area of the airship that is exposed to the sun in any given direction. The results show that the new method is more accurate and faster than the old one, allowing for a better and more reliable approximation of the sun facing surface area of the airship.
The possibility of using solar panels on the underside of the airship was also discussed. However, further research is needed to explore the feasibility of this option.

11.1 Conclusion

Solar radiation reaching a solar-powered airship is a very important factor for the possible flight duration, route, and speed. Therefore, having a realistic and accurate radiation model is crucial for the airship pathfinder software. The new model for assessing solar radiation reaching an airship has demonstrated greater accuracy and realism compared to the previous version. By taking into account the effects of weather conditions, the new model provides a better estimation of the solar power that can be generated by the airship. This allows the route optimization software to consider local weather phenomena and provide more accurate solar power values for a given route. The pathfinder will now adjust the travel altitude of the airship depending on the cloud height and try to circumnavigate the clouds to maximize the solar power output. Once the batteries are fully charged, the route finder will switch from a more solar-efficient route to a more direct one, as overproduction of solar power to charge the batteries is no longer required.
Combining this new model with the new solar area calculation makes it possible to calculate the solar power produced by the airship in different flight configurations. This can further be made even more precise by using the efficiency curves of different solar panels to calculate the power output for different wavelengths. This approach allows for the consideration of shifts in the light spectrum caused by atmospheric effects.
The idea of placing solar panels on the underside of the airship was also discussed and could be an interesting idea for increasing the solar power output of the airship as it has the ability to provide relatively high solar power output when flying over clouds or snow.

11.2 Outlook

Future transport of goods and people must be more sustainable than it is today. Airships have the potential to be almost entirely powered by solar energy, making them strong candidates for a sustainable alternative to airplanes and ships.

In recent years, interest in airships has grown, with many companies now working on new airship designs. This led to promising new scientific research in the field of airships, their viability, and their potential as a sustainable mode of transport. While the idea of airships as a mode of transport is not new and has already been tested in the past, the new improvement in solar panel technology makes them an interesting candidate for a more sustainable future.

Efficiently powering an airship using solar energy is crucial for its success. By using the newly presented model for the calculation of solar radiation reaching an airship, it is now possible to more accurately calculate the solar power output of the airship for different routes and weather conditions. The inclusion of weather data in the radiation calculation allows for a more realistic simulation of the solar power output of the airship. The effectiveness of this approach can already be seen in the new results of the airship Pathfinder software. Now it actively tries to avoid clouds or fly above them in order to maximize the solar power output of the airship. Once the batteries are fully charged, the route finder switches from a solar-efficient route to a more direct one, as overproduction of solar power in order to charge the batteries is no longer required.

The current implementation, however, suffers from a high runtime and could be further optimized and made faster by the use of more extensive caching and parallelization. The data provided by the new radiation calculation can also be used to optimize the number of solar panels and their placement on the airship. The best type of solar panel could differ depending on the route and weather conditions. For instance, airships operating in different regions might benefit from different solar panel configurations to maximize energy output.

Once new test flights of solar-powered airships are conducted, the data from these flights could be used to validate the simulation results. Currently, companies like LTA Research and Euro Airship are working on new airship designs and are planning to conduct test flights in the near future. Euro Airship is even planning a global circumnavigation using their solar-powered airship called "Solar Airship One" in 2026.

Airships have great potential in a future dominated by the need for more sustainable modes of transport. It will be fascinating to see what new kinds of improvements research into carbon-neutral transportation will bring in the coming years.

References

[1] *Global Modeling and Assimilation Office (GMAO) (2015), MERRA-2 inst3_3d_asm_Np: 3d,3-Hourly, Instantaneous, Pressure-Level, Assimilation, Assimilated Meteorological Fields V5.12.4, Greenbelt, MD, USA, Goddard Earth Sciences Data and Information Services Center (GES DISC)*. DOI: 10.5067/QBZ6MG944HW0. URL: https://disc.gsfc.nasa.gov/datasets/M2I3NPASM_5.12.4/summary (visited on 09/20/2024).

[2] "Transport and environment report 2022". In: *EEA Report* (May 2023).
. ISSN: 1977-8449. DOI: 10.2800/47438.

[3] *Freightos: Shipping from Germany to the US*. Dec. 2024. URL: https://www.freightos.com/shipping-routes/shipping-from-germany-to-the-us/ (visited on 12/15/2024).

[4] *Airshipsonline: R34 "The Record Breaker"*. URL: https://www.airshipsonline.com/airships/r34/index.html (visited on 11/27/2024).

[5] Veit Götz. "Batteriesimulation und Optimierung für Solarzeppeline". In: *Informatik 10* (2022), p. 10. URL: https://www10.cs.fau.de/publications/theses/2022/Bachelor_G%C3%B6tzVeit.pdf.

[6] Andrew T. Young. "Air mass and refraction". In: *Appl. Opt.* 33.6 (Feb. 1994), pp. 1108–1110. DOI: 10.1364/AO.33.001108. URL: https://opg.optica.org/ao/abstract.cfm?URI=ao-33-6-1108.

[7] C. Emde et al. "The libRadtran software package for radiative transfer calculations (version 2.0.1)". In: *Geoscientific Model Development* 9.5 (2016), pp. 1647–1672. DOI: 10.5194/gmd-9-1647-2016. URL: https://gmd.copernicus.org/articles/9/1647/2016/.

[8] J. Hocking J. Vidot. *Note on RTTOV unit conversions for gases, clouds and aerosols*. Nov. 2022. URL: https://nwp-saf.eumetsat.int/site/download/documentation/rtm/docs_rttov12/rttov_gas_cloud_aerosol_units.pdf (visited on 11/27/2024).

[9] Martin A. Green et al. "Solar cell efficiency tables (Version 63)". In: *Progress in Photovoltaics: Research and Applications* 32.1 (2024), pp. 3–13. DOI: https://doi.org/10.1002/pip.3750. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pip.3750. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pip.3750.

[10] Martin A Green. "Solar cell fill factors: General graph and empirical expressions". In: *Solid-State Electronics* 24 (1981).
, pp. 788–789. ISSN: 00381101. DOI: 10.1016/0038-1101(81)90062-9.

[11] Michael Jockisch. "Entwicklung einer Interface-Software und Visualisierung in Python für eine Atmosphären-Strahlungstransfermodellsoftware für Fernerkundungsanalysen". In: (2012). URL: https://www.th-rosenheim.de/fileadmin/fakultaeten/ing/Bilder/02_Labore/Mess-_und_Regelungstechnik/Abschlussarbeiten/Entwicklung-einer-Interface-Software-und-Visualisierung-in-Python-fuer-eine_Atmosphaeren-Strahl.-Jockisch_M.pdf.

[12] Javier Gurrola-Ramos, Oscar Dalmau, and Teresa E Alarcón. "A Residual Dense U-Net Neural Network for Image Denoising". In: *IEEE Access* 9 (2021), pp. 31742–31754. DOI: 10.1109/ACCESS.2021.3061062.

Acronyms

EQE external quantum efficiency

HPC High Performance Computing

MLP Multi-Layer Perceptron

OMP Open Multi-Processing

LibRadtran Library for Radiative transfer

stderr Standard Error

stdout Standard Output

sza solar zenith angle