UPPSALA
UNIVERSITET

IT kDV 25 011

Degree Project 15 credits

June 2025

Precise Positioning for Underwater Vessels

Axel Hallsenius

Bachelor's Programme in Computer Science

UPPSALA
UNIVERSITET

Precise Positioning
for Underwater Vessels

Axel Hallsenius

Abstract

The objective of this project is to identify and analyze software sources of positional discrepancies, with a focus on map projections and coordinate conversion. To achieve this, a program was developed that visualizes the differences between two methods for converting points between coordinate systems. The program was designed to translate between a local coordinate system and geodetic coordinate system. The implementation was carried out in the C programming language using established geospatial transformation formulas.

Two positioning algorithms were implemented: an accurate algorithm based on Gauss–Krüger formulas using Universal Transverse Mercator (UTM) grids, and a simplified, naive method that neglects the Earth's curvature, serving as a baseline for comparison.

To facilitate visual analysis, a Graphical User Interface (GUI) was developed, enabling users to display and compare the output of both transformation methods against the backdrop of a map of the Earth in the Mercator projection.

The discrepancies were found to likely be of a lesser degree than that of the naive method. The program demonstrated, however, that small inacurracies can compound over larger distances.

Faculty of Science and Technology

Uppsala University, Uppsala

Supervisor: Joel Åström Reviewer: Jingwei Dong

Examiner: Johannes Borgström

Acknowledgements

I'd very much like to thank

*My Subject Reviewer **Jingwei Dong** for helping me through the process of writing this thesis report. Your guidance for this thesis was instrumental to its completion.*

*My colleagues at Saab for making me feel welcome in a new office and in a new town, especially **Joel Åström** and **Emil Fritz** for your steady and active guidance throughout this project. Our discussions were vital for its success.*

***Marie Vilhelmsson** for believing in me as a candidate for this project, and allowing me to come work with you for a while.*

*My friends and family for supporting me, especially **Johan Hallsenius**, my dad, and **Mirjam Andersson**, my girlfriend, for proofreading.*

***Jonas Jonsson** for welcoming me, a stranger, into your home, during my time in Karlskoga.*

Table of Contents

1 Background

Saab, a defense technology company, has a vested interest in accurately determining the position of its deployed underwater vessels. These vessels are unmanned and operate either under remote control or autonomously. According to Saab's technical documentation, the position of a deployed vessel is concurrently tracked by both the vessel itself and an independent observer system. However, over time, a discrepancy arises between the two systems' estimations of the vessel's location.

Saab is investigating whether this divergence may be attributed to inaccuracies in the transformation between local and geodetic coordinate systems. Geodetic coordinates, commonly expressed as latitude and longitude, are denoted in this report as (φ, λ).

The observer system receives periodic position updates from the vessel. Between these updates, it employs predictive calculations to estimate the vessel's current position. This estimation process is illustrated in Figures 1 and 2 (on page 5).

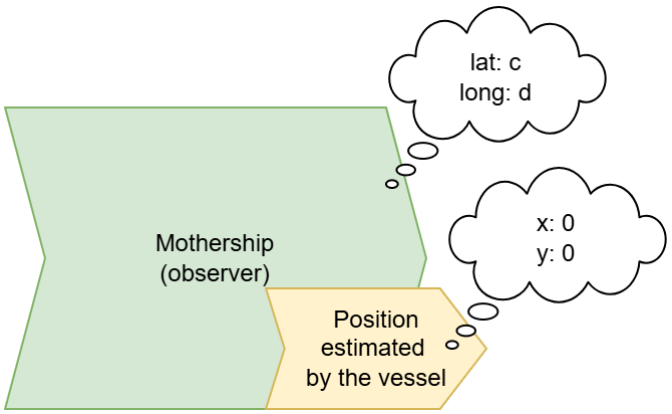

Figure 1: *At the point of release, the "Mothership" knows the coordinates of the vessel because it knows its own coordinates. The vessel measures its position from this point of release.*

The goal of this project is to compare and visualize paths made by the vessel and the observer, compare different ways of navigation, and in this way eliminate sources for error in the navigation system. Inputs will be provided as $(meter, meter)$-vectors, as horizontal directions for the vessel to travel. In the code, these are referred to as `move_order`, and are generated randomly using a pseudo-random generator.

For Saab the highest priority is getting the observer system and the vessel to agree on a position, after which, the speed of execution takes the next highest priority. There are limits for memory usage, but minimizing it is a secondary concern to those previously mentioned.

1.1 Projections and Geodesy

To enable correct navigation, accurate projection formulas need to be used. To understand why navigation using a flat map is challenging, the reader is encouraged to read up on map projections. The following section provides an overview of map projections, geoids and ellipsoids as they pertain to this project.

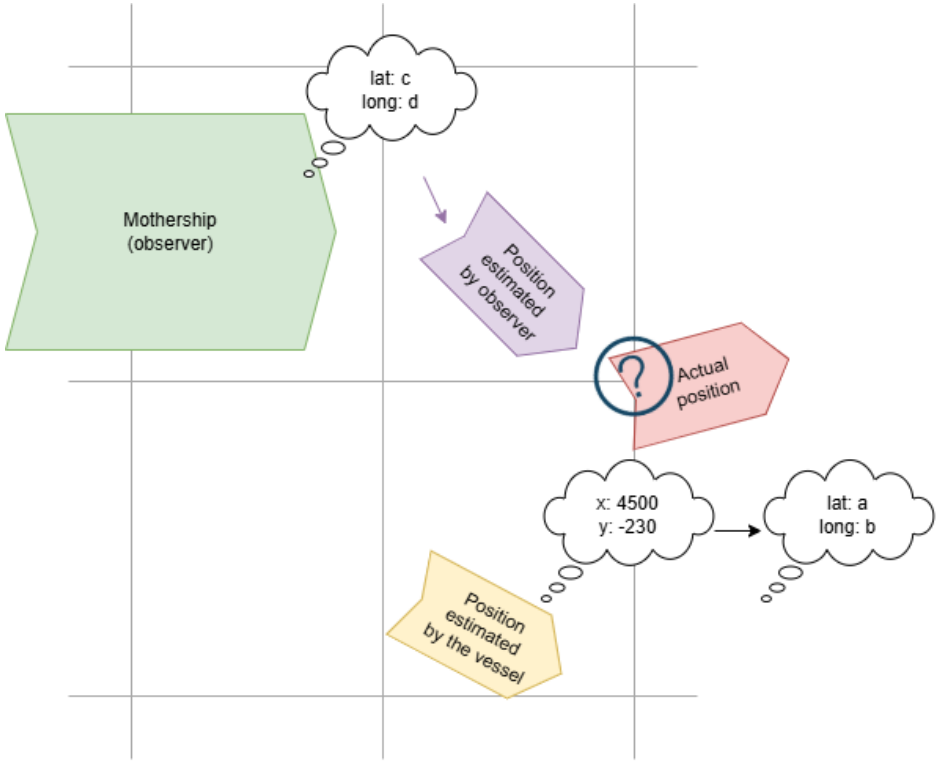

Figure 2: *The position estimated by the observer and the vessel itself have deviated. It is unknown whether either are correct. They could both be incorrect.*

When plotting a straight course, it is common for the plotted line to appear curved when looking at a flat map. This occurs due to the earth's spheroid geometry. Flat maps display a projection of the actual shape of the Earth (which will be discussed further down) onto a flat plane (the map).

1.1.1 Map Projections

To generate flat maps of locations on the earth, the planet (or sections thereof) need to be projected onto a plane. The fundamental challenge of map projections is commonly illustrated by peeling an orange, and then trying to get the peel into a rectangular shape. To remedy this, cartographers use different projections in order to distort the actual shape of the Earth to fit into a flat map. Different projections suit different tasks. They have their pros and cons as the nature of projection necessitates some distortions compared to reality. These distortions can carry errors into navigation if the navigator is not careful.

1.1.2 Mercator Projection

The Mercator projection preserves angles by stretching meridians to compensate for the stretching of the parallels [1]. The process can be visualized as stretching a sphere to the straight side of a cylinder (not the round top or bottom), then unrolling the cylinder, as illustrated in Figure 3(on page 6). Figure 4 (on page 6) illustrates how instead of stretching, it's also possible to contract meridians between certain parallels in order to flatten a map.

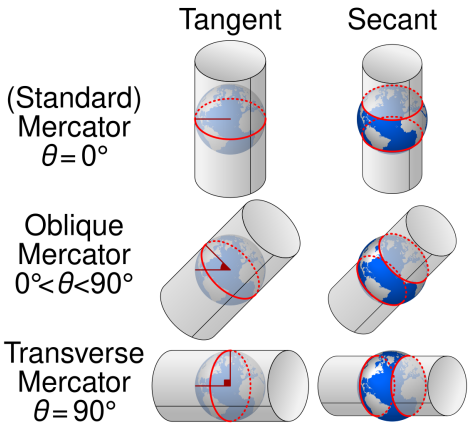

Figure 3: *Illustration of Mercator projections*

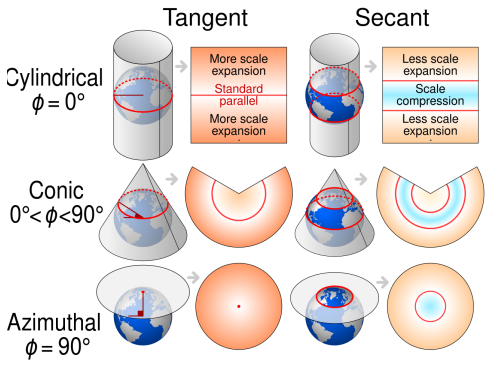

Figure 4: *Illustration of the difference between tangent and secant projections*

1.1.3 Great Circle Line

An arc-measurement is a method by which the navigator determines the length $\triangle$G of a great circle. This was done from ancient until modern times to measure distances on a sphere (in the navigator's case; Earth) [2]. When looking at a flat map, during a long flight for example, the path taken looks curved, while it is in reality a straight arc line. The website www.greatcirclemap.com [3] has been used to provide illustration of this effect. A great circle line between two cities as seen in Figure 5 is distorted on the Mercator projection of Google Maps, as seen in Figure 6 (on page 7).

Nowadays the shape of the Earth is much better known, and it is now common to use a rotational ellipsoid as a model for the earth, which approximates its level surface within $\pm 100m$ [2].

Figure 5: *A straight line (in red) on the Earth's surface as visualised from above using www.greatcirclemap.com*

Figure 6: *www.greatcirclemap.com lets you switch view. The same line as in Figure 5 is illustrated over Google Maps, showing the distortion of the line over a flat projection*

1.1.4 Geoids, Ellipsoids and WGS84

The field of geodesy concerns creating approximations of the shape of the Earth, a *geoid* being such an approximated model. For navigation, the shape most commonly used is a level ellipsoid. There are several standards for such Earth ellipsoids. The two most used are GRS 1980, the international standard, and WGS84 that is used by the GPS satelite navigation system [4]. This project utilizes the WGS84 ellipsoid.

From time to time, recommended ellipsoidal models are updated and released by the International Union of Geodesy and Geophysics (IUGG) [2]. The shape of the three dimensional ellipsoid is described by two parameters, the semi-major axis a and the semi-minor axis b, though b is generally not used, but replaced by a flattening factor $f = \frac{a-b}{a}$ instead [2].

Figure 7 (on page 8) illustrates the imperfect roundness of the earth to an exaggerated degree, by multiplying its so called geoid undulations (how it deviates from a subtrahend model) by 10000.

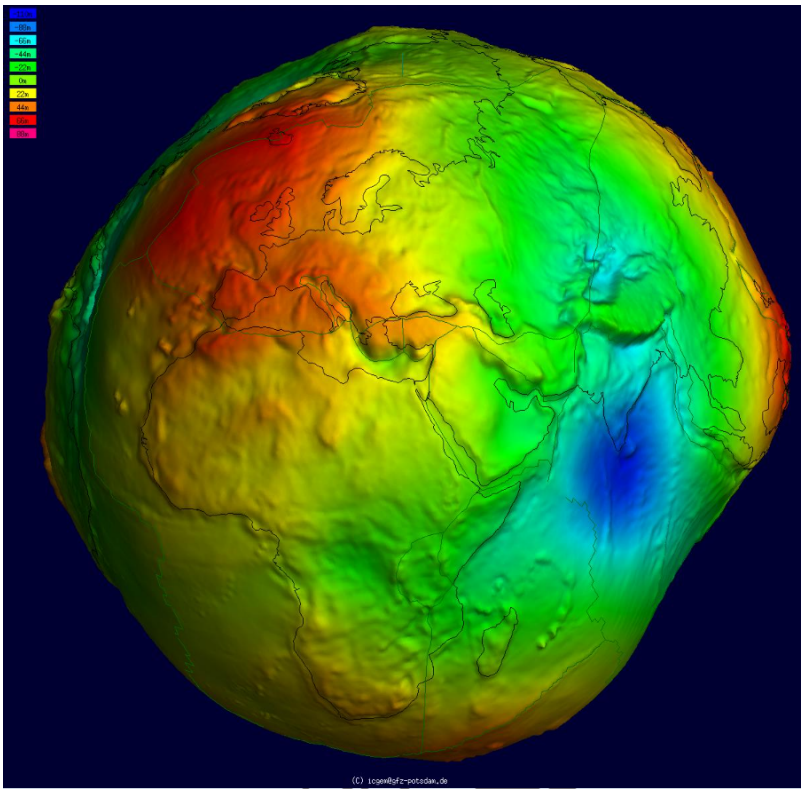

Figure 7: *An exaggerated geoid model that illustrates the imperfect shape of the Earth*

1.1.5 Gauss-Krüger Projection

This project utilizes Gauss-Krüger (GK) projection formulas. The source for these formulas is the Swedish mapping, cadastral and land registration authority (Lantmäteriet), and it can be found in reference [5]. Please refer to appendix A for the formulas, and to 5.0.3 for an explanation of how they are implemented and used.

1.1.6 UTM

Universal Transverse Mercator (UTM) is a widely used reference system in which the earth is divided into 60 longitudinal zones that are 6° wide each. These zones are divided in two along the equator, for the north and south hemispheres. The *false northing* is defined to be 0m on the northern hemisphere, and 10 000 000m on the southern hemisphere. The *false easting* is always 500 000m [4]. This ensures that coordinates in the southern hemisphere can be made positive [4]. This grid is a derivative of the general GK projection [6].

1.2 Related Work

1.2.1 GPS and GNSS

GPS is well known to fail in providing time and location below the surface. According to Gunnar Taraldsen, Tor Arne Reinen and Tone Berg [7], this is because the electromagnetic signals from the satellites are dampened by the water. Studies [8] provide acoustic localization as a

functioning alternative. This is a technology where surface (or temporarily surfacing) vessels acquires GPS data, before diving and providing this data via some of acoustic ping to other submerged vessels.

These types of acoustic alternatives, by their definition make sound, which is undesirable to vessels that may want to hide its position from a third party. This is why acoustic alternatives are disregarded in this case. This project will rely on geodesic and cartographical methods for navigating.

1.2.2 Mapping and Projection Software

Graphic Information System (GIS) is a term used for systems (primarily software) that is used for analyzing geospatial data, map production and visualization. There are multiple open source distributions available, for example GRASS GIS and Marble.

There are also command line utilities like PROJ that are focused on coordinate conversion and projections. PROJ does the coordinate conversion for several big projects, like GRASS GIS. It is also used by many state actors in geodesy, like Lantmäteriet and the United States Geological Survey [9].

Generic Mapping Tools (GMT) is a tool suite that can be used to produce maps of the Earth, or regions thereof, to show the plotted positions in relation to geographic landmarks. This tool contains options for generating maps using many different map projections. According to an article in Geochemistry, Geophysics, Geosystems [10], GMT is used ubiquitously in Earth and ocean sciences, and by tens of thousands of scientists around the world.

PROJ or GMT would be great fits for this project, were it not for the facts that Saab prefers to use in-house software when possible, and that in this case the developing process itself is a key part to the debugging of their existing software.

2 Problem formulation

The objective of this project is to identify and analyze software sources of positional discrepancies, with a focus on map projections and coordinate conversion. To achieve this, a program with these requirements has been implemented:

The program should be able to:

- View vessel paths and positions in relation to a map
- Translate movements of points in a cartesian grid into accurate (φ, λ) positions on a map projection
- Compare this accurate path to a less accurate one
- Tests to check for function correctness
- Obey roughly estimated memory and CPU constraints (in the 2020's laptop ballpark)
- The positioning needs to be accurate to within a meter.

And if possible, provide some *nice-to-haves*:

- A GUI with an options menu
- Ability to set starting position
- Zooming and panning the map
- Make the positioning work in polar zones
- Make the positioning work over the date-line
- Through the development process help solve the bug (or possible user error) in Saab's software

While the movement of the vessel depthwise does influence navigation, vertical movement was considered to have minor impact and thus left outside of the scope of this project.

3 Program Architecture and User Interface

3.1 SDL3, SDL3_gfx and SDL3_image

The C library ”Simple Directmedia Layer” (SDL, specifically SDL3 [11]) is used as a base upon which to build the graphical application. SDL3 is used to produce a window and a drawing loop that is used to display the map, grid and interactive menus (explained below) for the application. SDL3_gfx is a library that adds some functionality to SDL3. In this project, this library primarily serves to draw circles.

SDL3_image is a small extension to SDL3 that allows for easier manipulation of image files as a part of the project. In this project, it's used to correctly load the image of the earth into a texture.

As demonstrated in listing 6, the `SDL_RenderLine` function is used to draw a route in between coordinates. This draws a path between the coordinates converted from the `move_order` into geodetic coordinates. These (φ, λ) coordinates are given as an argument to this function in the form of a `point_geodetic`. The function then converts the geodetic point to a position value in the window.

3.2 ImGui

ImGui is a C++ library that provides an easy way to make option menus and other GUI components. This will be used to provide options for the user to view certain elements of the program separately, or at the same time, according to preference. The user can drag the ImGui menus across the simulation window in case the menu covers something important.

The *ImGui* menu is defined by a bracket and a call to `ImGui::Begin()` as a ”window”, referring not the whole application window, but to the black rectangle containing the menu 2. The options within are defined using *ImGui*'s built in functions for check boxes 3), radio menus 4 (for the XOR selection of an option in e.g. an `enum`.), and plain text 1. These options change variables that are instantiated in the *Setup* portion of the code. The menu allows the user to change the value of these variables (mostly boolean), in order to activate or deactivate the visibility of corresponding parts of the simulation. In Figure 8 (on page 12) the ImGui, as used in this project, can be seen.

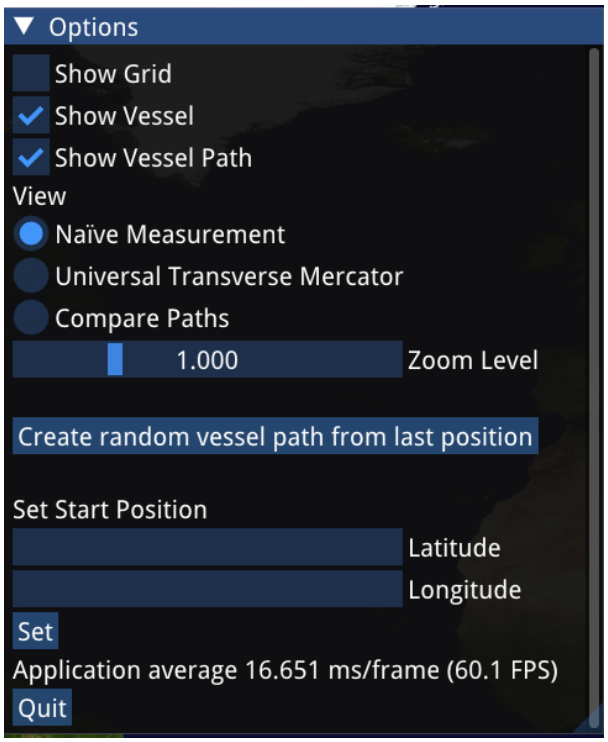

Figure 8: *The ImGui menu*

3.3 Control Flow Charts

This section describes which function calls which, and how the program arrives at a solution. Figure 9 (on page 13) explains how the paths are rendered and drawn on the map. Figure 10 (on page 14) illustrates the process of translating a `move_order` into positions on a map. Each position is translated again into pixel values. This happens every frame, since the correct pixel value changes depending on where the user has zoomed or panned.

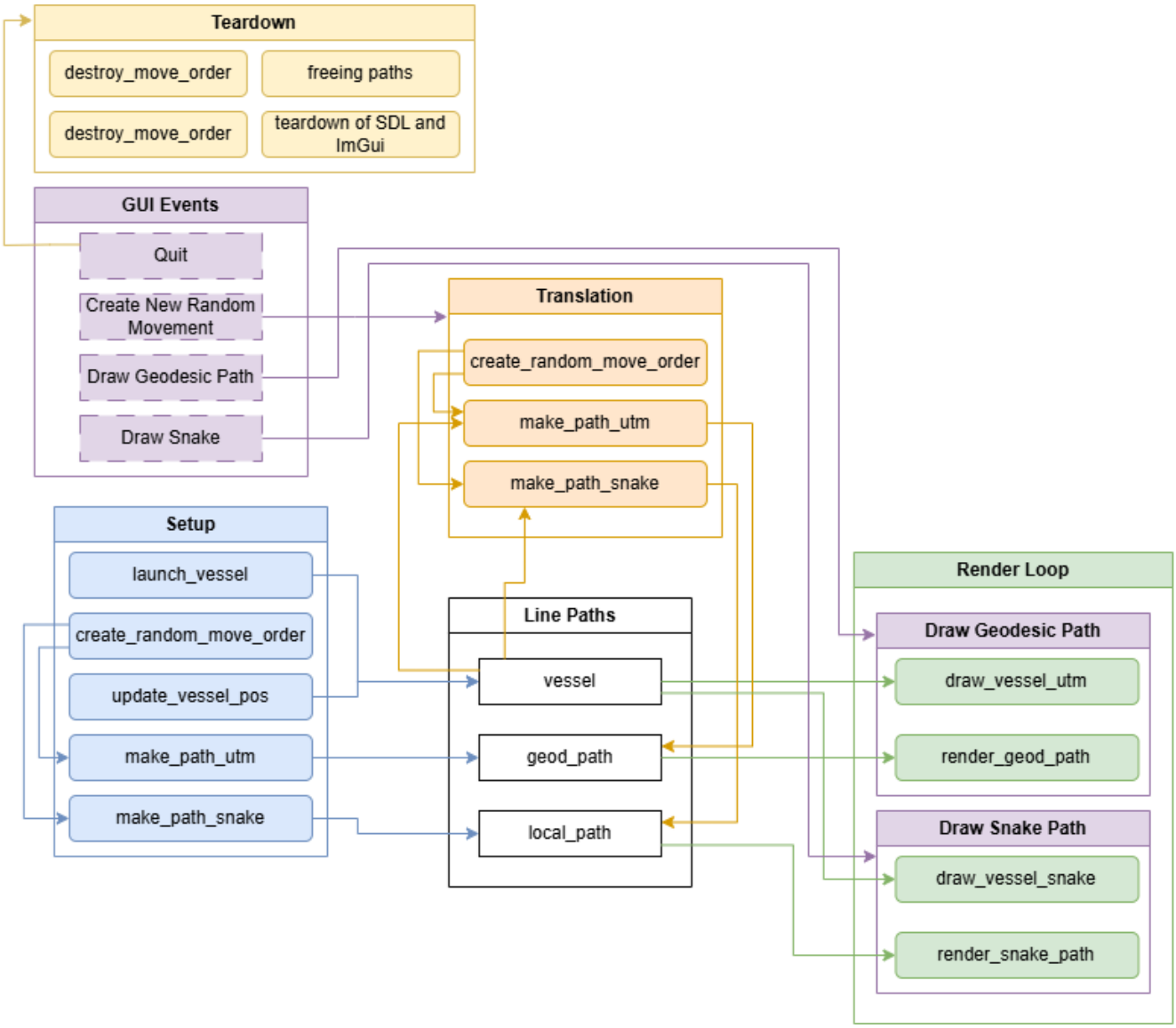

Figure 9: *Control flow when rendering paths. Render Loop runs every frame. Translation is only done when the user presses the "Create New Random Move" button, and a new move order is received.*

Figure 10 describes the control flow when translating points from a move order to geodetic points (φ, λ). What is not visible in Figure 10 is that `utm_grid_to_geod` is run twice per coordinate. The first time is to acquire the correct `utm_zone`. On this first pass, the answer can be a bit off. This is because the point of the zones is that the same set of (x, y) grid coordinates mean different locations depending on in which zone the target is. The second pass uses the new zone to calculate the location correctly.

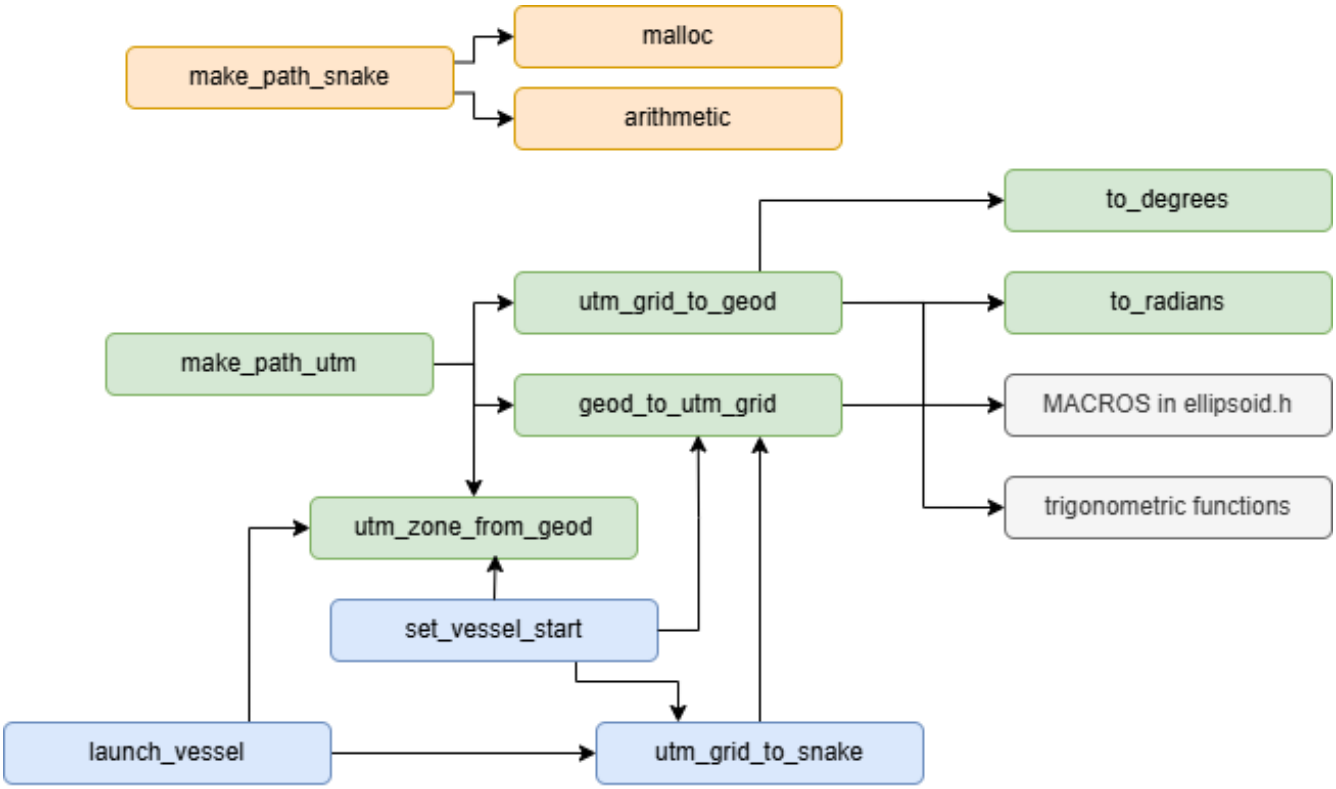

Figure 10: *Translating arrays of (x, y) movements into arrays of geodetic positions and cartesian positions, respectively*

Moving from one zone to another provides a challenge. If the algorithm ignored zone boundaries, the UTM funcitons would provide little improvement to the ”snake” algorithm. Since coordinates in a transverse Mercator grid are not unique, but can represent places in multiple different zones, the zone must also be specified. When converting points in a UTM projection into (φ, λ) points, the function `utm_grid_to_geod` takes both (x, y) coordinates and a zone as arguments.

3.4 GUI

Two dots of different colors represents the same vessel positioned using the two different methods. One using the naive approach (explained in 3.5), and the other, the more accurate approach using the UTM projection from an ellipsoid (explained in 1.1.4). The program also has the ability to draw routes in the color of the dot corresponding to the same projection. There is a menu to provide the user with a way to configure the visualization during runtime.

In `vessel_tracker.cpp`, the `main` function defines variables and calls functions defined in the other source files. `main` has 3 parts:

- **Setup**
  This is where most resources are allocated, and functions that only need to run once are run.
- **Loop**
  This code is run each frame. Objects that need to be drawn (everything visible) need to be here. For example; the program needs to keep track of where it's supposed to draw a line for this particular frame, as circumstances could change the next. The map needs to be rendered for every frame, so as to ”draw over” old lines.
- **Cleanup**
  This is where most allocated resources are freed, so as to prevent memory leaks. This part of the main function is triggered when the user pushes the ”Quit” button.

3.5 Navigation Formulas

The translation functions convert move orders, which are a set of (x, y) meter vectors into their respective target coordinate systems (illustrated in Figure 11). Some illustrations of the different coordinate systems follow. Note that these illustrations are not to scale, and included purely for demonstrative purposes. The move orders are randomly generated.

To enable precise coordinate conversion, the program implements Gauss-Krüger's formulas. Movement is done in a projected plane, as depicted in Figure 13 (on page 16). The new (x, y) position in the projected plane is translated into (φ, λ) to place it on a map. For more information on how these functions interact, refer to Figure 10 (on page 14).

For comparison, the program implements a baseline for what happens if one was to take a very naive approach. The naive mode will simplify the world to an (x, y) plane. The way this method saves coordinates is illustrated in Figure 12 (on page 16). The only real challenge here is to transform these (x, y) coordinates into relevant (x, y) values for display in a window. In the code, this is referred to as the ”snake” projection, because of it's similarity to the old Nokia phone game *Snake*.

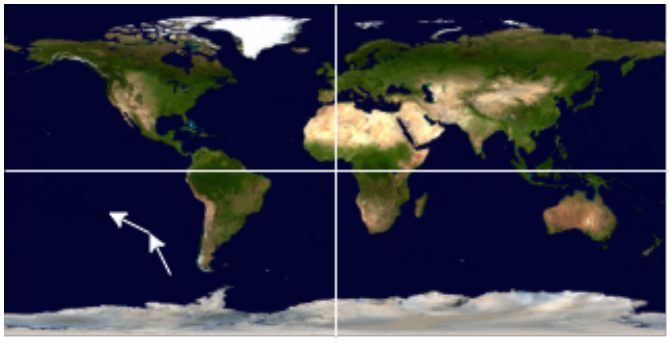

Figure 11: *A short move order. A set of vectors, set intended as (x, y) meters from the previous waypoint.*

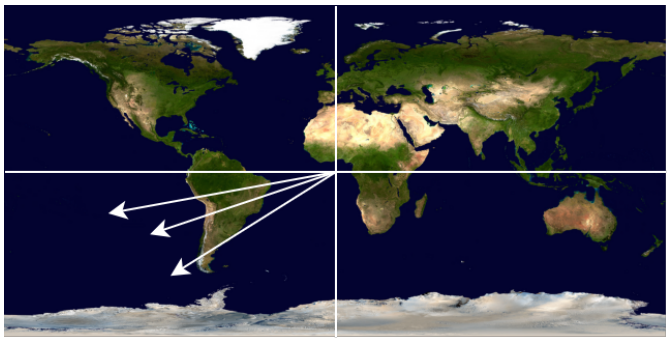

Figure 12: *The "snake" coordinate system. Coordinates are translated into vectors to the waypoints from the origin.*

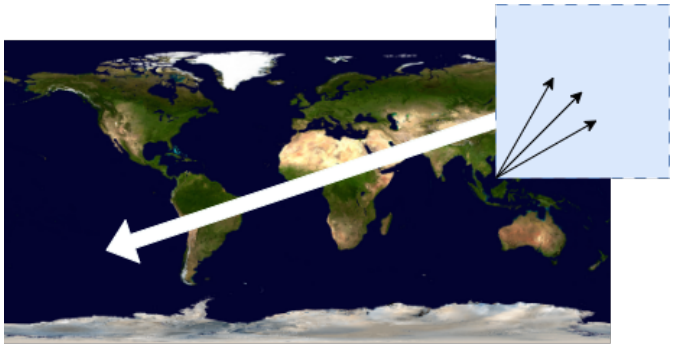

Figure 13: *Illustration of how (x, y) meters in a UTM grid works. These positions are saved as vectors from the false origin of the local UTM grid.*

3.6 Testing

This project utilizes the framework *GoogleTest*, documentation for which is available here. Included is an example of a tests from `tests/test_map.cpp` 1. The tests have been split up into separate files, but are all called by the `main()` function in `test_all.cpp`.

The conversion between (φ, λ) degrees, and (x, y) meters (inside an UTM projection) is tested using results from a tool by the US National Geodetic Survey (NGS) called NCAT [12] as reference. The USGS is an American governmental organization, which provides information on ecosystems, land use, energy and mineral resources, natural hazards, water use and availability, and updated maps and images of the Earth's features to the public [13].

The program has the ability to generate a set of random waypoints. In code this is called a `move_order`, illustrated in Figure 11. These waypoints are (x, y) coordinates, defined as a 2D-vector of meters from the end of the previous waypoint. The random value is set to be between -100 to 100. This number is then multiplied by a scale factor set by a macro in `vessel\_tracker.cpp`.

This project utilizes *GoogleBenchmark* to check the performance of certain functions, the git repository for which can be found here. The repository also contains documentation. Listing 12 is an example of a benchmark.

4 Results

4.1 Build and Program Execution Instructions

A link to this project's repository is available here. Should the reader wish to build the program from source, the instructions below or in the repository can be followed to do so. There are currently only instructions for UNIX platforms available. While the program should work on most UNIX based platforms, it has only been tested on Linux.

A prerequisite for building is `cmake`. This is a build tool used to compile the code into an executable. In order to acquire the project, use `git`.

Downloading:

```
git clone --recurse-submodules
↪ https://github.com/axelhallsenius/bachelor_thesis.git
```

Building:

For convenience, a `Makefile` is provided, so the user may simply `cd` into the `code` and there run

```
make
```

which will run the needed `cmake` commands for you.

In order to run the projects tests, simply run

```
make test
```

There is also a `clean` command provided in the `Makefile`, should the user wish to rebuild or remove the build directory. For this run

```
make clean
```

To run benchmarks:

```
make bench
```

4.2 Explanation of Files, Datastructures and Functions

- **vessel_tracker.cpp**
  This file started out as a template from the ImGui codebase, but was heavily modified.
- **map.c/h:**
  In `map.c` the conversion formulas between coordinate systems is defined. See listing 7 for the datastructures in `map.h`.
- **ellipsoid.h:**
  This is where all the ellipsoidal constants are defined. There are also some other macros defined here that are useful for the projection calculations in map.c.
- **vessel.c/h:**
  This module contains functions which convert the input path (which is randomized as demonstrated in listing 9).
- **draw.c/h:**
  `draw.c` defines functions for interacting with the graphical portion of the code. Imports `vessel.h`.

4.3 Test Results

Tests converting local (x, y) into (φ, λ) is asserted to be within $1 * 10^5$ of a degree of the reference result. Likewise, tests checking transfer of zones have the same deviation limit. Tests for converting (φ, λ) into (x, y) in a UTM projection are asserted to be within one centimeter of deviation from the reference. All tests pass

5 Discussion

There are multiple tests for positioning accuracy when converting coordinates in local UTM grids to (φ, λ) and vice versa. One such is seen in listing 1. Using the data from NCAT [12], the test cases report sub-centimeter accuracy for position conversion. Likely, this source uses the same (or similar) formulas.

5.0.1 Path Conversion Algorithms

In order to provide a course for the vessel to travel that can be translated onto the map using the two different methods, there is a function made to randomize a series of horizontal movements, given as (x, y) coordinates, as can be seen in listing 9. The number of steps in this path, and the size of them, can be edited using macros. By default these are set as: `ORDER_LEN` $= 1000$ and `ORDER_SCALE` $= 10$. This means the array of moves generated is 1000 entries long, and with a scale in the order of hundreds of meters (between -1 to $1km$ in both x and y directions).

5.0.2 Positioning Using the ”Snake Method”

The ”Snake Method” is used for demonstrative purposes, to show how navigation according to a flat map of the earth will accumulate errors overtime. This model disregards the curvature of the Earth, and defines it as a rectangle of
40 075 016m · 20 004 000m.

5.0.3 Positioning Using Gauss-Krüger Formulas

The code in listing 13 defines the function to translate (x, y) movement into movement into (φ, λ). The reverse operation is also possible, as seen in listing 15. Test results (as seen in section 4.3) show accuracy for both of these functions are well within acceptable parameters.

Figures 14 (on page 20), 15 (on page 21) and 16 (on page 22) are screenshots of the program running, showing the map, the vessels and the menu. The program draws the naive path in green and the properly converted path in red. In Figure 14 (on page 20) we see the paths of a vessel that started at a (φ, λ) of (0, 0), where the Equator and Prime Meridian meet. Here the paths differs only marginally. The deviation is not visible at the scale of the Gulf of Guinea.

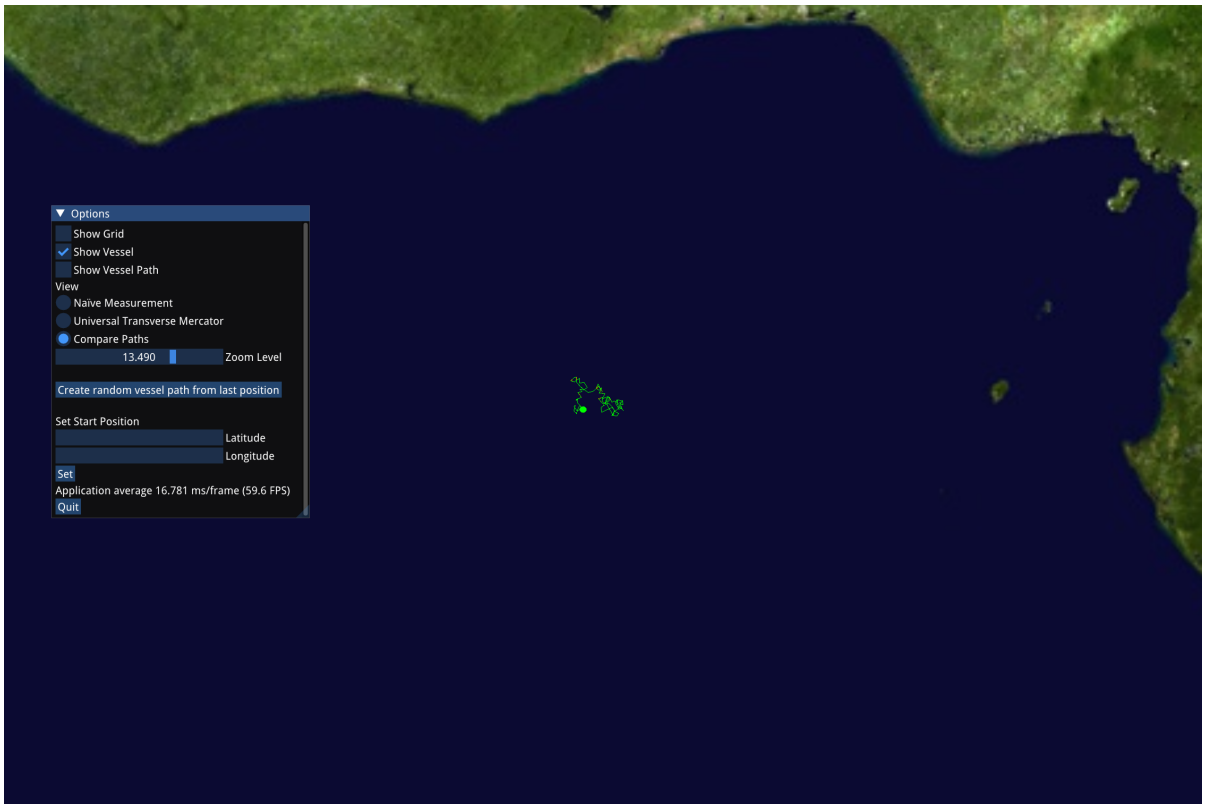

Figure 14: *At this latitude and scale distortion is minimal*

Even when moving in areas with relatively low distortion, the difference eventually start to show, as seen in Figure 15 (on page 21). These paths are the last in a series of 25 iterations of creating new random paths. The distance between the waypoints in these paths are in the order of tens of meters, so the difference between the paths here would be in the order of meters. An error in several meters is well above acceptable tolerance.

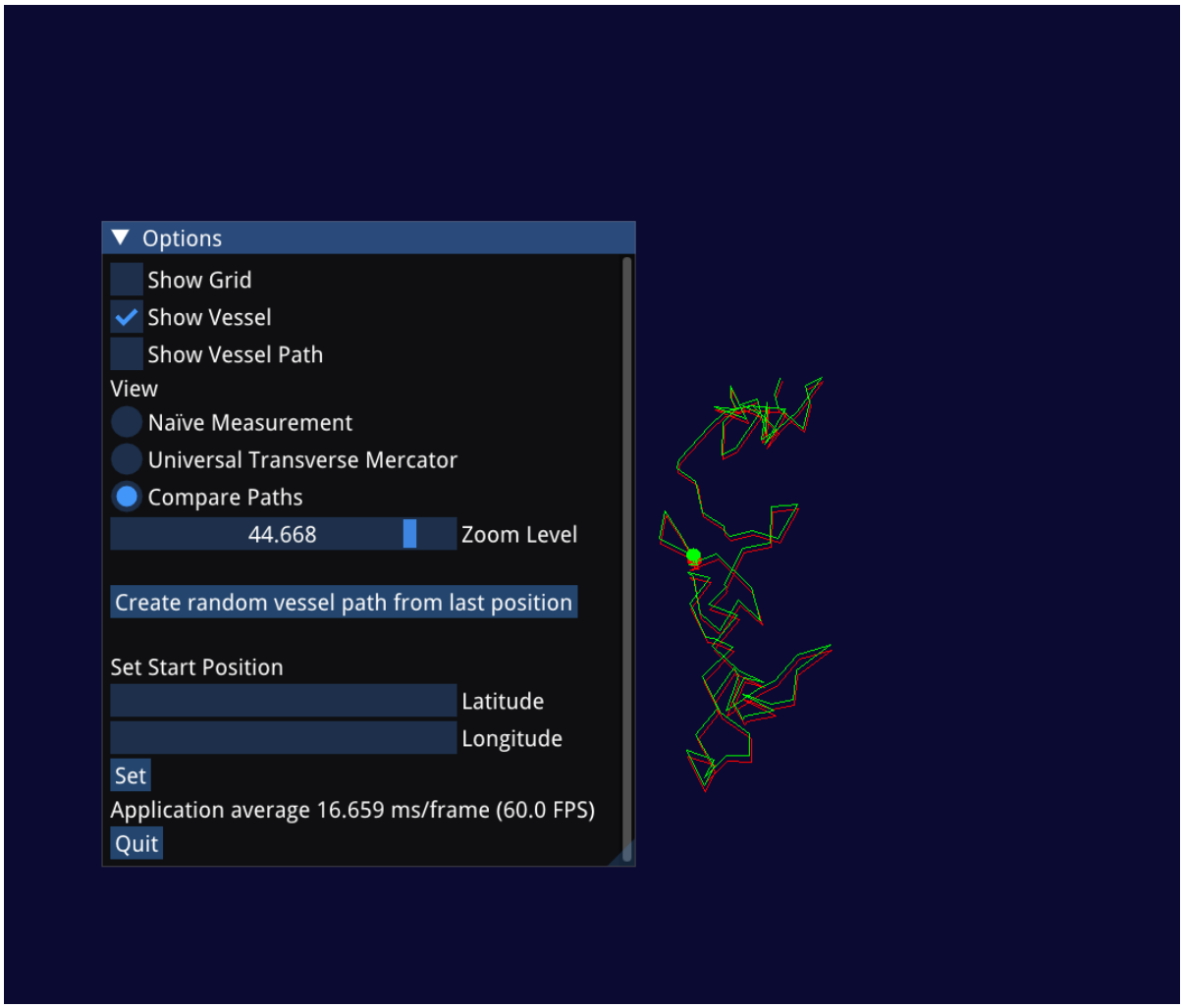

Figure 15: *Visible difference after 25000 waypoints*

Figure 16 (on page 22) demonstrates that when moving at higher or lower latitudes, the difference becomes obvious. The naive path looks to be less distorted on the map, but since the map itself is distorted here, the green path is instead distorted from reality.

Figure 16: *Obvious difference at Baltic latitudes*

5.1 Comparing Resource Metrics

The respective translations have been benchmarked (for how this was done, refer to section 3.6). The results of the benchmark can vary by hardware. On the hardware available for this project, the results are visible in benchmark result C.1. Measurements on the user's systems can be achieved by running `make bench`. Benchmark results indicate that, as expected, the Gauss-Krüger implementation demonstrates significantly higher computational overhead as compared to the ”Snake” implementation.

5.2 Paths Not Taken

Understanding how map projection formulas are used and how to implement them in code took a substantial portion of the allotted time for the project.Several functions and data structures were implemented but later discarded due to time constraints. For example, more general functions, the purpose of which was to make calculations for *a given* projection instead of a set one (UTM), were considered and halfway implemented, but discarded in the interest of time. The value of using other projections, rather than a globally used standard, good enough for much more advanced projects, was deemed to be low.

5.3 Future Work

5.4 Optimization

There has yet to be a proper time complexity analysis of the algorithms in this project. There are likely functions that could have lower time complexities. An optimization pass was intended, but not completed due to time constraints.

Rendering the paths, the map and the ImGui menu every frame is resource intensive and with the program's current functionality, can be considered a waste of resources. While re-rendering every frame is not necessary for the program as it is currently, in the future, one may wish to introduce live moving elements.

5.4.1 Polar Navigation and Exceptions

At time of writing, the projection code is only implemented between 86N-80S. The UTM coordinate system is commonly used together with the Universal Polar Stereographic (UPS) projection. When these projections are used in conjunction, UTM is used between 84° N and 80° S. In UPS, the Arctic and Antarctic are mapped two separate projections that are used above 86° and below 80° respecitvely. Due to time constraints, the UPS was left unimplemented in this project.

In UTM, there are also exceptions in zoning made around the coast of Norway, and around Svalbard. In this project, this detail is ignored. This is however acceptable for the purposes of navigation in this program, since it should not impact the navigation formulas.

5.4.2 Vincenty's Formulas

Vincenty's formulas are geodesic ellipsoid solutions designed to conserve space and execution time in computers. Vincenty in his original article [14], gives two formulas: the ”direct” and the ”inverse” ”solution of geodesics on the ellipsoid”. He also clams that non-iterative solutions consume more space, and that some of the contemporary solutions were even slower than the solution in his article.

There is a branch in the git repository in which Vincenty's formulas for navigation are implemented, and halfway integrated. Due to lack of time for integration with the GUI and testing, Vincenty's formulas were left on the drawing table.

5.4.3 Quaternion Navigation

Upon having the problem described, an immediate thought was to use unit quaternions for positioning and rotation, as is common in programming videogames and robotics. This implementation could have made use of an ellipsoidal coordinate system like the Celestial Reference Frame (CRF) or similar, and place points (and rotation) on the Earth's surface. Substantial time and a very exact geoid model would likely be needed to make this work.

5.4.4 Eliminating Other Potential Error Sources

Several potential error sources (not discussed in this report) for the position of an underwater vessel can be imagined. Water depth and salinity could impact the vessel's sensors, and record false readings for its course and speed. Underwater currents could also cause deviations in course and heading that could impact accuracy of its reported position. These sources are out of scope for this project, as it is focused solely on software.

5.4.5 Further Streamlining of the Build Process

The build process is currently in an acceptable state, but there are improvements that could be made to the `CMakeLists.txt` files to make it much faster, and less prone to having to be rebuilt.

5.5 State of the Art

5.5.1 GNSS-Aucustic Positioning

Milind Fernandes, Soumya Ranjan Sahoo, and Mangal Kothari discuss in their paper[15] the use of acoustic signals between underwater vehicles to provide accurate navigation through a sort of signaling cluster could hold promise. Acoustic signals were briefly mentioned and rejected in section 1.2.1, however, there may still be ways this can be useful. A vessel that doesn't want its location to be known could perhaps make use of many buoys spread over a large area, or long battery life vessels semi-permanently patrolling stochastic routes. These would rise to get GPS signal and dive to send an acoustic signal, like illustrated in paper [15].

5.6 Conclusion

A GUI was made, and made clear the difference that projection errors can make. Tests show the implemented algorithms are well within the bounds of acceptable accuracy.

This project demonstrates that even small projection inaccuracies can compound over long paths, resulting in deviations of several meters. While Saab's issue is likely not as severe as the naive model, these findings suggest projection or zone miscalculations remain a credible candidate.

References

[1] P. Osborne, *The Mercator Projections*, 2013.

[2] W. Torge and J. Müller, *Geodesy (4th Edition)*. De Gruyter, 2012.

[3] M. Englund, *Great Circle Map*, Available at https://www.greatcirclemap.com/ (2025/04/22).

[4] J. Iliffe and R. Lott, *Datums and map projections for remote sensing, GIS, and surveying*, Second edition. Scotland, United Kingdom: Whittles Publishing, 2008.

[5] *Gauss Conformal Projection (Transverse Mercator), Krüger's Formulas*, Lantmäterigatan 2C, Gävle, Sweden: Lantmäteriet Informationsförsörjning, Geodetiska utvecklingsenheten, Aug. 1, 2008.

[6] M. Hooijberg, *Practical Geodesy, Using Computers*. Springer Berlin, Heidelberg, Dec. 21, 2011.

[7] G. Taraldsen, T. A. Reinen, and T. Berg, "The Underwater GPS Problem," in *OCEANS 2011 IEEE - Spain*, 2011, pp. 1–8. DOI: 10.1109/Oceans-Spain.2011.6003649.

[8] H.-P. Tan, R. Diamant, W. K. Seah, and M. Waldmeyer, "A Survey of Techniques and Challenges in Underwater Localization," *Ocean Engineering*, vol. 38, no. 14, pp. 1663–1676, 2011. DOI: 10.1016/j.oceaneng.2011.07.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801811001624.

[9] PROJ contributors, *PROJ coordinate transformation software library*, Open Source Geospatial Foundation, 2025. [Online]. Available: https://proj.org/en/stable/users.html#users.

[10] P. Wessel, J. F. Luis, L. Uieda, R. Scharroo, F. Wobbe, W. H. F. Smith, and D. Tian, "The Generic Mapping Tools Version 6," *Geochemistry, Geophysics, Geosystems*, vol. 20, no. 11, pp. 5556–5564, 2019. DOI: 10.1029/2019GC008515.

[11] *SDL wiki entry on SDL_Texture*. [Online]. Available: https://wiki.libsdl.org/SDL3/SDL_Texture.

[12] US National Geodetic Survey, *NGS Coordinate Conversion and Transformation Tool (NCAT)*, Apr. 22, 2025. [Online]. Available: https://www.ngs.noaa.gov/NCAT/.

[13] USGS, *USGS: Who We Are*. [Online]. Available: https://www.usgs.gov/about/about-us/who-we-are.

[14] T. Vincenty, "Direct and Inverse Solutions of Geodesics of the Ellipsoid with Application of Nested Equations," *Survey Review XXII*, Apr. 1975.

[15] M. Fernandes, S. Ranjan Sahoo, and M. Kothari, "Cooperative Localization for Autonomous Underwater Vehicles – a comprehensive review," *arXiv e-prints*, Jul. 2023. DOI: 10.48550/arXiv.2307.06189.

Appendices

A Gauss-Krüger Formulas

Constants

a	: semi-major axis of the ellipsoid
f	: flattening of the ellipsoid
e^2	: first eccentricity squared
φ	: geodetic latitude, positive north
λ	: geodetic longitude, positive east
x	: grid coordinate, positive north
y	: grid coordinate, positive east
λ_0	: longitude of the central meridian
k_0	: scale factor along the central meridian
$\delta\lambda$	: difference $\lambda - \lambda_0$
FN	: false northing
FE	: false easting

All angles are expressed in radians. Please note that the x-axis is directed to the north and the y-axis to the east.

The following variables are defined out of the ellipsoidal parameters a and f:

$$e^2 = f(2 - f)$$

$$n = \frac{f}{2 - f}$$

$$\hat{a} = \frac{a}{1 + n}\left(1 + \frac{1}{4}n^2 + \frac{1}{64}n^4 + \dots\right)$$

Conversion from geodetic coordinates (φ, λ) to grid coordinates (x, y)

Compute the conformal latitude φ^*:

$$\varphi^* = \varphi - \sin\varphi\cos\varphi\left(A + B\sin^2\varphi + C\sin^4\varphi + D\sin^6\varphi + \dots\right)$$

$$\begin{aligned} A &= e^2 \\ B &= \frac{1}{6}\left(5e^4 - e^6\right) \\ C &= \frac{1}{120}\left(104e^6 - 45e^8 + \dots\right) \\ D &= \frac{1}{1260}\left(1237e^8 + \dots\right) \end{aligned}$$

Let $\delta\lambda = \lambda - \lambda_0$ and define:

$$\xi' = \arctan\left(\frac{\tan\varphi^*}{\cos\delta\lambda}\right), \quad \eta' = \operatorname{atanh}\left(\cos\varphi^* \cdot \sin\delta\lambda\right)$$

$$\begin{aligned} x = k_0\hat{a}\big(&\xi' + \beta_1 \sin 2\xi' \cosh 2\eta' + \beta_2 \sin 4\xi' \cosh 4\eta' \\ &+ \beta_3 \sin 6\xi' \cosh 6\eta' + \beta_4 \sin 8\xi' \cosh 8\eta' + \dots\big) + FN \\ y = k_0\hat{a}\big(&\eta' + \beta_1 \cos 2\xi' \sinh 2\eta' + \beta_2 \cos 4\xi' \sinh 4\eta' \\ &+ \beta_3 \cos 6\xi' \sinh 6\eta' + \beta_4 \cos 8\xi' \sinh 8\eta' + \dots\big) + FE \end{aligned}$$

Conversion from grid coordinates (x, y) to geodetic coordinates (φ, λ)

$$\xi = \frac{x - FN}{k_0\hat{a}}, \quad \eta = \frac{y - FE}{k_0\hat{a}}$$

$$\begin{aligned} \xi' = \xi &- \delta_1 \sin 2\xi \cosh 2\eta - \delta_2 \sin 4\xi \cosh 4\eta \\ &- \delta_3 \sin 6\xi \cosh 6\eta - \delta_4 \sin 8\xi \cosh 8\eta - \dots \\ \eta' = \eta &- \delta_1 \cos 2\xi \sinh 2\eta - \delta_2 \cos 4\xi \sinh 4\eta \\ &- \delta_3 \cos 6\xi \sinh 6\eta - \delta_4 \cos 8\xi \sinh 8\eta - \dots \end{aligned}$$

Series Coefficients

$$\begin{aligned} \delta_1 &= \frac{1}{2}n - \frac{2}{3}n^2 + \frac{37}{96}n^3 - \frac{1}{360}n^4 + \dots \\ \delta_2 &= \frac{1}{48}n^2 + \frac{1}{15}n^3 - \frac{437}{1440}n^4 + \dots \\ \delta_3 &= \frac{17}{480}n^3 - \frac{37}{840}n^4 + \dots \\ \delta_4 &= \frac{4397}{161280}n^4 + \dots \end{aligned}$$

Final Formulas

$$\varphi^* = \arcsin\left(\frac{\sin\xi'}{\cosh\eta'}\right), \quad \delta\lambda = \arctan\left(\frac{\sinh\eta'}{\cos\xi'}\right)$$

$$\lambda = \lambda_0 + \delta\lambda$$

$$\begin{aligned} \varphi = \varphi^* + \sin\varphi^* \cos\varphi^* \big(&A^* + B^* \sin^2\varphi^* + C^* \sin^4\varphi^* \\ &+ D^* \sin^6\varphi^* + \dots\big) \end{aligned}$$

Inverse Series Coefficients

$$A^* = e^2 + e^4 + e^6 + e^8 + \dots$$
$$B^* = -\frac{1}{6}(7e^4 + 17e^6 + 30e^8 + \dots)$$
$$C^* = \frac{1}{120}(224e^6 + 889e^8 + \dots)$$
$$D^* = -\frac{1}{1260}(4279e^8 + \dots)$$

True and False Origins:

- True Origin
  The *true origin* is at the intersection of the Prime Meridian and the Equator.
- False Origin
  The *false origin* is a point set usually to the south and west of the projected area, which makes it possible for all coordinates in the area to be positive. The coordinates of the *false origin* are called the *false easting* and *false northing*.

B Code Listings

Table of Code Listings

```
TEST(MeterToGeodetic, AngstromCrossroad){
  point_geod geop;
  point_geod ans;
  ans.deg_long = 17.6447886229;
  ans.deg_lat = 59.8399342516;
  point_tm_grid gridp;

  utm_zone zone = utm_zone_from_geod(ans);

  gridp.y = 6636543.595;
  gridp.x = 648206.154;
  geop = utm_grid_to_geod(gridp, zone);

  EXPECT_NEAR(ans.deg_long, geop.deg_long, 0.005);
  EXPECT_NEAR(ans.deg_lat, geop.deg_lat, 0.005);
}
```

Listing 1: *Example of a test using the GoogleTest framework*

```
 {
    ImGui::Begin("Options");

    /*
     * Contents of window
     */

    ImGui::End();
}
```

Listing 2: *ImGui commands defining a window*

```
ImGui::Checkbox("Show Vessel", &show_vessel);
```

Listing 3: *Defining checkboxes in ImGui*

```
if(ImGui::RadioButton("Naïve Measurement", view == snake_vessel)) {
  view = snake_vessel;
}
if(ImGui::RadioButton("Universal Transverse Mercator",
  view == utm_vessel)) {
  view = utm_vessel;
}
if(ImGui::RadioButton("Compare Paths", view == compare)) {
  view = compare;
}
```

Listing 4: *Defining radio menus in ImGui*

```
ImGui::SliderFloat(
    "Zoom Level",
    &zoom,
    0.10f,
    100.0f,
    "%.3f",
    ImGuiSliderFlags_Logarithmic
    );

ImGui::Text("\n");
```

Listing 5: *Examples of other things in ImGui: plain text, and a slider (used for zoom)*

```
void render_geod_path(canvas_t *canvas, point_geod *path, int len){
  if (len > 1){
    SDL_FPoint curr = geod_to_pixels(canvas->dst_rect, path[0]);
    SDL_FPoint prev;

    for(int i = 1; i < len; i++){
      prev = curr;
      curr = geod_to_pixels(canvas->dst_rect,path[i]);

      SDL_RenderLine(canvas->renderer, prev.x, prev.y, curr.x, curr.y);
    }
  }
}
```

Listing 6: *Rendering lines in between waypoints*

```c
typedef struct {
  int hemisphere; //north is 0 and south is 1
  int c_meridian; //central meridian of the zone
  // (in degrees from the prime meridian)
} utm_zone;

typedef struct {
  double a; //semi major axis
  double f; // 1/f : flattening factor.
  // (this f != 1/f. 1/f is calculated)
} tm_ellipsoid;

typedef struct{
  double deg_lat;
  double deg_long;
} point_geod;

typedef struct {
  double x;
  double y;
} point_local;

typedef struct{
  double x;
  double y;
} point_tm_grid;


```

Listing 7: *Datastructures in* `map.h`

```c
typedef struct {
  SDL_Renderer *renderer;
  SDL_FRect    *dst_rect;
}canvas_t;

typedef struct {
  int len;
  point_local *deltas;
} move_order_t;

typedef struct {
  //where the vessel was launched. lat/long
  point_geod start_geod;

  // where the vessel is currently at. lat/long
  point_geod pos_geod;
  utm_zone zone;

  point_local start_snake;
  point_local pos_snake;
} vessel_t;
```

Listing 8: *Datastructures in* `vessel.h`

```c
move_order_t *create_random_move_order(int len, int scale){
  move_order_t *order = malloc(sizeof(move_order_t));
  order->deltas = malloc(len * sizeof(point_local));
  order->len = len;

  double dx;
  double dy;

  for (int i = 0; i < len; i++){
    dx = ((SDL_rand(200) - 100));
    dy = ((SDL_rand(200) - 100));
    dx = dx * scale;
    dy = dy * scale;

    point_local p = {dx, dy};

    order->deltas[i] = p;
  }
  return order;
}
```

Listing 9: *How a* `move_order` *is randomized. A* `move_order` *is a datastructure containing an array of* `pos_local`*. The* `pos_local` *provides the* Δx, Δy *for distance (in meters) and direction.*

```c
point_geod *make_path_utm(point_geod start, move_order_t *order){
  point_geod *path = malloc(order->len * sizeof(point_geod));

  point_geod tmp_geod;
  utm_zone zone;
  point_tm_grid prev;
  point_tm_grid curr = geod_to_utm_grid(start);
  path[0] = start;
  for (int i = 1;i < order->len;i++) {
    zone = utm_zone_from_geod(path[i-1]);
    prev = curr;
    curr.x = prev.x + ((order->deltas)[i].x);
    curr.y = prev.y + ((order->deltas)[i].y);

    tmp_geod = utm_grid_to_geod(curr, zone);
    zone = utm_zone_from_geod(tmp_geod);
    curr = geod_to_utm_grid(tmp_geod);

    path[i] = utm_grid_to_geod(curr, zone);
  }

  return path;
}
```

Listing 10: *A `move_order` is translated into an array of (ϕ, λ) values (`point_geod`). The zone is recalculated from the new degree position. In order to get the exact (ϕ, λ) correct, the conversion is run again, but now using the correct zone.*

```c
point_local *make_path_snake(point_local start, move_order_t *order){
  point_local *path = malloc(order->len * sizeof(point_local));

  point_local prev;
  point_local curr = start;
  path[0] = curr;
  for (int i = 1;i < order->len;i++) {
    prev = curr;
    curr.x = prev.x + ((order->deltas)[i].x);
    curr.y = prev.y + ((order->deltas)[i].y);

    path[i] = curr;
  }

  return path;
}
```

Listing 11: *This function converts the local movements into an array of global positions (from the origin). It is not meant as a serious attempt at navigation, but as a reference to compare the other function to.*

```
1
2  static void BM_zone_transfer(benchmark::State& state) {
3    point_geod geop;
4    point_geod ans;
5    point_tm_grid gridp;
6    ans.deg_lat = -42.1634034242;
7    ans.deg_long = 146.6015625000;
8    gridp.x = 467086.882;
9    gridp.y = 5332004.203;
10   point_geod other_zone = {-37.37000, 120.230000};
11
12
13
14   for (auto _ : state) {
15     //this code gets timed
16     utm_zone zone = utm_zone_from_geod(other_zone);
17     geop = utm_grid_to_geod(gridp,zone);
18     zone = utm_zone_from_geod(ans);
19     geop = utm_grid_to_geod(gridp,zone);
20   }
21 }
22 BENCHMARK(BM_zone_transfer);
23
```

Listing 12: *Example of a benchmark using GoogleBenchmark*

```
point_geod utm_grid_to_geod(point_tm_grid p, utm_zone z){
  point_geod coords;
  double fn = UTM_FN_N;
  if (z.hemisphere == HEM_S){
    fn = UTM_FN_S;
  }

  double xi = (p.y - fn) /
    (UTM_CMER_SCALE * UTM_A_HAT);

  double eta = (p.x - UTM_FE) /
    (UTM_CMER_SCALE * UTM_A_HAT);

  double xi_prime = xi -(
    (D1 * sin(2 * xi) * cosh(2 * eta)) -
    (D2 * sin(4 * xi) * cosh(4 * eta)) -
    (D3 * sin(6 * xi) * cosh(6 * eta)) -
    (D4 * sin(8 * xi) * cosh(8 * eta)));

  double eta_prime = eta - (
    (D1 * cos(2 * xi) * sinh(2 * eta)) -
    (D2 * cos(4 * xi) * sinh(4 * eta)) -
    (D3 * cos(6 * xi) * sinh(6 * eta)) -
    (D4 * cos(8 * xi) * sinh(8 * eta)));

  double conf_lat = asin(sin(xi_prime) / cosh(eta_prime));

  double delta_lambda = atan(sinh(eta_prime) / cos(xi_prime));

  double rad_lat =
    conf_lat + sin(conf_lat) * cos(conf_lat) * (
    A_STAR +
    (B_STAR * pow(sin(conf_lat), 2)) +
    (C_STAR * pow(sin(conf_lat), 4)) +
    (D_STAR * pow(sin(conf_lat), 6)));

  double lambda = to_radians(z.c_meridian) + delta_lambda;
  coords.deg_long = to_degrees(lambda);
  coords.deg_lat = to_degrees(rad_lat);

  return coords;
}
```

Listing 13: *Converting positions in a local grid into exact* (φ, λ) *coordinates*

```
utm_zone utm_zone_from_geod(point_geod p){
  utm_zone z;
  z.hemisphere = (p.deg_lat < 0);

  // zone number (1 .. 60)
  int zone_number = (int)floor((p.deg_long + 180.0)/6.0) + 1;
  if (zone_number < 1) zone_number = 1;
  if (zone_number > 60) zone_number = 60;

  // central meridian = -183deg + 6deg zone_number
  z.c_meridian = -183 + zone_number * 6;

  return z;
}
```

Listing 14: *Acquiring the correct central meridian and hemisphere*

```
point_tm_grid geod_to_utm_grid(point_geod p){
  utm_zone zone = utm_zone_from_geod(p);
  double fn = UTM_FN_N;
  if (zone.hemisphere == HEM_S){
    fn = UTM_FN_S;
  }

  double phi = to_radians(p.deg_lat);
  double lambda = to_radians(p.deg_long);

  const double e2 = E2;
  const double e4 = e2*e2, e6 = e4*e2, e8 = e4*e4;

  const double A =  (0.5*e2 + 5.0/24*e4 + 3.0/32*e6 + 281.0/5760*e8);
  const double B =  (5.0/48*e4 + 7.0/80*e6 + 697.0/11520*e8);
  const double C =  (13.0/480*e6 + 461.0/13440*e8);
  const double D =  (1237.0/161280*e8);

  double conf_lat =
    phi
    - A*sin(2*phi)
    + B*sin(4*phi)
    - C*sin(6*phi)
    + D*sin(8*phi);

  double delta_lambda = lambda - to_radians(zone.c_meridian);

  double xi_prime =
```

```
29      atan(tan(conf_lat) /
30           cos(delta_lambda));
31
32    double eta_prime =
33      atanh(cos(conf_lat) *
34            sin(delta_lambda));
35
36    double sum1 = (
37      (B1 * sin(2.0 * xi_prime) * cosh(2.0 * eta_prime)) +
38      (B2 * sin(4.0 * xi_prime) * cosh(4.0 * eta_prime)) +
39      (B3 * sin(6.0 * xi_prime) * cosh(6.0 * eta_prime)) +
40      (B4 * sin(8.0 * xi_prime) * cosh(8.0 * eta_prime)));
41
42    double sum2 = (
43      (B1 * cos(2.0 * xi_prime) * sinh(2.0 * eta_prime)) +
44      (B2 * cos(4.0 * xi_prime) * sinh(4.0 * eta_prime)) +
45      (B3 * cos(6.0 * xi_prime) * sinh(6.0 * eta_prime)) +
46      (B4 * cos(8.0 * xi_prime) * sinh(8.0 * eta_prime)));
47
48    point_tm_grid coords;
49    double easting = (UTM_CMER_SCALE * UTM_A_HAT * (eta_prime + sum2)) + UTM_FE;
50    double northing = (UTM_CMER_SCALE * UTM_A_HAT * (xi_prime + sum1)) + fn;
51    coords.x = easting;
52    coords.y = northing;
53    return coords;
54  }
```

Listing 15: *Translating degree positions into positions in an UTM grid*

```
TEST(ZoneTransfer, EquatorCrossing){ //sub cm accuracy
  point_geod geop;
  point_geod ans;
  point_tm_grid gridp;
  ans.deg_lat = -42.1634034242;
  ans.deg_long = 146.6015625000;
  gridp.x = 467086.882;
  gridp.y = 5332004.203;
  point_geod other_zone = {-32.5400, 130.7800};

  utm_zone zone = utm_zone_from_geod(other_zone);
  geop = utm_grid_to_geod(gridp,zone);
  zone = utm_zone_from_geod(ans);
  geop = utm_grid_to_geod(gridp,zone);

  EXPECT_NEAR(ans.deg_lat, geop.deg_lat, 0.00001);
  EXPECT_NEAR(ans.deg_long, geop.deg_long, 0.00001);
}
```

Listing 16: *Example of a test where the start point is in another zone to the target. Several tests of this type are made to cover edge cases.*

C Benchmark Results

```
Run on (4 X 3500 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x2)
  L1 Instruction 32 KiB (x2)
  L2 Unified 256 KiB (x2)
  L3 Unified 3072 KiB (x1)
Load Average: 1.15, 1.28, 1.02
***WARNING*** CPU scaling is enabled, the benchmark real time measurements may
 ↪  be noisy and will incur extra overhead.
------------------------------------------------------------------
Benchmark                        Time             CPU   Iterations
------------------------------------------------------------------
BM_geod_to_meter               287 ns          286 ns      2373944
BM_meter_to_geod               668 ns          667 ns      1010019
BM_zone_transfer               697 ns          696 ns      1003230
BM_zone_transfer               690 ns          688 ns      1033470
BM_random_move_order         16026 ns        15958 ns        43834
BM_make_path_snake            9079 ns         9034 ns        73111
BM_make_path_utm            929796 ns       927734 ns          762
```

Benchmark Result C.1: *`make_path_snake` and `make_path_utm` uses the output from `create_random_move_order`. The input to these benchmarks is 1000 waypoints long.*