# A framework for integrating data, models and visualisation tools to understand fish migration pattern

Sebastian Frøyen,
Torkel Kårstad Nes

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 2019

Western Norway
University of
Applied Sciences

**Abstract**

This thesis generate trajectories of fish migrations by running simulations of temperature and depth values from Digital Storage Tags. The data is compared with data from ocean general circulation models in order to find Global Positioning System (GPS) locations of fish. Integrating this solution with a visual analytics tool allows the user to analyse fish trajectories through visualisation and manipulation.

**Acknowledgements**

We would like to thank our supervisors at Western Norway University of Applied Sciences (HVL), Harald Soleim, Atle Birger Geitung, and Daniel Patel for their guidance throughout this thesis. Their frequent feedback and assistance has helped us greatly. We would also like to thank Sam Subbey from the Insititute of Marine Research (IMR) for being a great external supervisor. Without his assistance, expertise and feedback, this thesis would not have been possible. Finally we would like to thank Vidar Lien, Bjørn Ådlandsvik, Kathrine Michalsen, Jon Albretsen and Solfrid Hjøllo for sharing knowledge, testing the framework, and helping us whenever necessary.

# Division of Work

Chapter 1 - Introduction - **Both**

- 1.1 Thesis Outline - **Nes**
- 1.2 Motivation - **Nes**
- 1.3 Available Data - **Nes**
- 1.4 Goal - **Nes**
- 1.5 Related Work - **Both**
- 1.6 Research Question - **Nes**
- 1.7 Research Method - **Nes**

Chapter 2 - Background - **Both**

- 2.1 Data and Models - **Both**
  - 2.1.1 Tag Data - **Nes**
  - 2.1.2 Ocean General Circulation Models - **Nes**
  - 2.1.3 NetCDF - **Both**
- 2.2 Simulation Application - **Both**
  - 2.2.1 .NET Framework - **Both**
  - 2.2.2 Python - **Froyen**
  - 2.2.3 Choosing .NET Framework - **Froyen**
- 2.3 Visualisation Application - **Both**
  - 2.3.1 Game Engines - **Both**
  - 2.3.2 Visualising Simulated Trajectories - **Froyen**
  - 2.3.3 Unity3D - **Both**
  - 2.3.4 Mapbox API - **Nes**
  - 2.3.5 Choosing Unity3D - **Both**

Chapter 3 - Design & Solution - **Both**

- 3.1 Application Design - **Nes**
  - 3.1.1 Simulation Design - **Froyen**
  - 3.1.2 Visualisation Design - **Nes**

- 3.2 Simulation Application - **Nes**
  - 3.2.1 Simulation Overview - **Nes**
  - 3.2.2 General Algorithm - **Both**
  - 3.2.3 Release Continuously Algorithm - **Both**
  - 3.2.4 Merge Algorithm - **Froyen**
  - 3.2.5 Comparison of Algorithms - **Froyen**
  - 3.2.6 Ending Simulated Trajectories - **Froyen**
  - 3.2.7 Grid Points Versus Latitude and Longitude - **Froyen**
  - 3.2.8 Validating Calculated Locations - **Froyen**
  - 3.2.9 Weighting - **Froyen**
  - 3.2.10 Speed - **Froyen**
  - 3.2.11 Static Data - **Froyen**
  - 3.2.12 Depth - **Froyen**
  - 3.2.13 Time Step - **Froyen**
  - 3.2.14 Implementation of Ocean Current - **Froyen**
  - 3.2.15 Choosing Optimal Ocean General Circulation Model - **Froyen**
  - 3.2.16 Reading NetCDF - **Froyen**
  - 3.2.17 DSTs used - **Froyen**
- 3.3 Visualisation Application - **Nes**

Chapter 4 - Results - **Both**

- 4.1 Research Method - **Nes**
- 4.2 Expert Survey - **Nes**
  - 4.2.1 Approach - **Nes**
  - 4.2.1 Results - **Nes**
- 4.3 Visualisation Application - **Nes**
- 4.4 Simulation Application - **Froyen**
  - 4.4.1 Ocean Current - **Froyen**
  - 4.4.2 DST 742 - **Froyen**
  - 4.4.3 DST 1664 - **Froyen**
  - 4.4.4 Switching Between Ocean General Circulation Models - **Froyen**

- 4.5 Performance - **Froyen**
  - 4.5.1 Time Consumption - **Froyen**
  - 4.5.2 Parallel Computing - **Froyen**
- 4.6 System Recommendations - **Froyen**

Chapter 5 - Discussion & Conclusion - **Both**

- 5.1 Discussion - **Nes**
  - 5.1.1 Algorithms - **Froyen**
  - 5.1.2 Time Consumption of the Algorithms - **Froyen**
  - 5.1.3 Visualisation Application - **Nes**
  - 5.1.4 Simulation Application - **Froyen**
  - 5.1.5 Switching Between Ocean General Circulation Models - **Froyen**
- 5.2 Conclusion - **Nes**
  - 5.2.1 Simulation Application - **Both**
  - 5.2.2 Visualisation Application - **Nes**
  - 5.2.3 Performance - **Froyen**
  - 5.2.4 Expert Opinions - **Nes**

Chapter 6 - Further Work - **Nes**

- 6.1 Simulation Application - **Nes**
- 6.2 Visualisation Application - **Nes**

# Contents

# List of Figures

# List of Tables

# Glossary

$\eta$ The x-coordinate in the ocean general circulation models.

$\sigma$ The z-coordinate in the ocean general circulation models.

$\xi$ The y-coordinate in the ocean general circulation models.

**Digital Storage Tags** Stores temperature and depth of fish in time intervals. This tag measures from the time the fish is released until it is recaptured.

**game engine** Development environment for the creation of video games.

**Ocean general circulation models** A model with a 3D grid of a limited area of the ocean containing data such as; temperature, depth and salinity..

**Scientific DataSet Lite** A cross platform library that can maipulate NetCDF files.

**Trajectory** A set of coordinates from the release location to the recapture location of a fish.

**WebGL API** A JavaScript API used to render 3D and 2D graphics in compatible browsers.

# Acronyms

**.exe** executable file.

**2D** two-dimensional.

**3D** three-dimensional.

**CLR** Common Language Runtime.

**CPU** central processing unit.

**DSTs** Digital storage tags.

**FCL** Framework Class Library.

**GPS** Global Positioning System.

**GPU** Graphics Processing Unit.

**IMR** Institute of Marine Research.

**RAM** random-access memory.

**SDSLite** Scientific DataSet Lite.

# Chapter 1

# Introduction

## 1.1 Thesis Outline

**Introduction** Chapter 1 provides a brief introduction to this thesis. The goal of the thesis is given along with the research questions. An overview of work related to the thesis is also presented.

**Background** Chapter 2 describes the background information needed for this thesis. Information about the technologies and programming languages that are used is provided.

**Design & Solution** Chapter 3 presents the design of the framework and describes the implemented solution.

**Results** Chapter 4 present the results and discuss the research method used in this thesis.

**Discussion & Conclusion** Chapter 5 gives a more detailed discussion about the results from the implemented framework and a conclusion on the research questions.

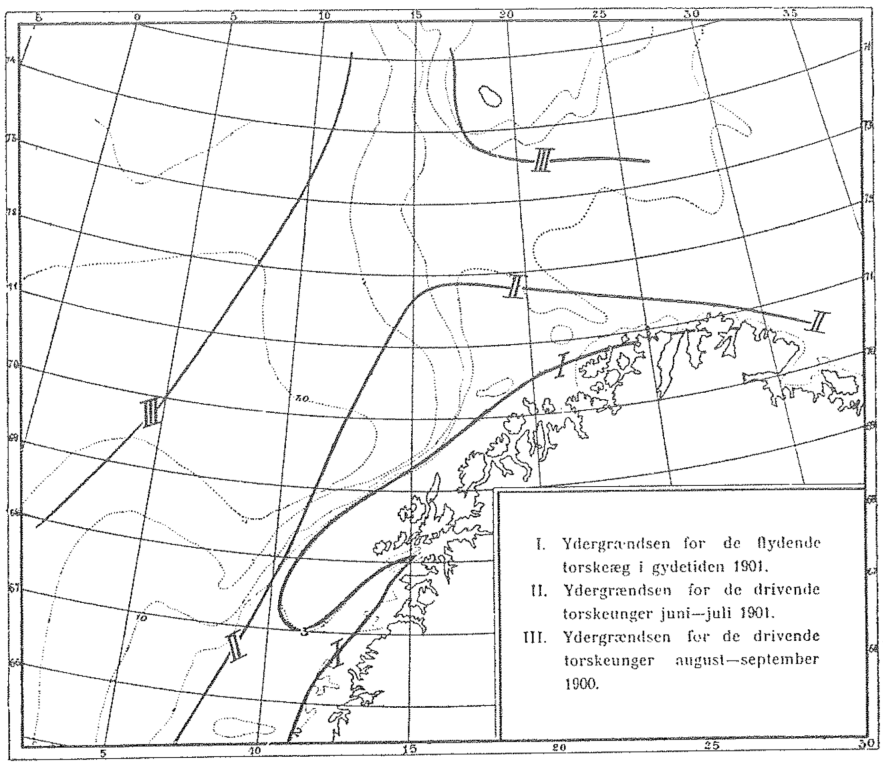
**Further work** Chapter 6 give a summary of improvements that can be implemented in further work.

## 1.2 Motivation

In 1878, a cod was captured in the waters around Spitsbergen with a fishing hook embedded into its flesh. This type of fishing hook was typically used for fishing in Lofoten, which strengthened the marine biologist Georg Ossian Sars' assumption from 1876. He believed that cod larvae drifted all the way from Vestfjorden

to Spitsbergen, but migrated back when it was time for spawning. Due to lack of observations, it was difficult to claim this with certainty.

In 1902, Johan Hjort showed that the outer borders of cod larvae was spread out according to their age, proving G.O. Sars' migration pattern correct [1]. The outer borders are shown in figure 1.1, where I) is the floating cod eggs, II) is the drifting of cod larvae in June - July, and III) is the drifting of cod larvae in August - September.




Udbredelsen af de flydende eg og torskeyngel til forskjellige aarstider.

Figure 1.1: Outer borders of cod larvae [1]

Hjort further performed a tagging program, tagging cod with silver buttons to the gill covers in Vestfjord during spawning season. The tagging program found seasonal migration patterns and differences between age groups and their geographical regions [2]. Since Hjort's program the Institute of Marine Research (IMR) have carried out tagging programs to better understand migration patterns of fish [3], and in 1996 they introduced Digital Storage Tags that store temperature and depth values of fish in time intervals. This tag type takes measurements from the time the fish is released until it is recaptured [4]. DSTs are still used and have gathered a lot of important data from fish for the IMR.

A second type of data, are those from ocean general circulation models. OGCMs were first developed in the 1970s. The model consist of latitude and longitude, and

associated variables such as; temperature, depth, and ocean current[5].

The IMR want a solution that simulate possible fish trajectories to potentially understand their migration patterns, but since data from DSTs does not include Global Positioning System (GPS) coordinates of fish locations it is difficult to determine where fish have traversed. DSTs and OGCMs contain temperature and depth in their data sets, and the IMR have both DSTs and OGCMs that correspond in time, but not a solution that combines the available data to recreate fish trajectories.

An earlier solution [6] managed to derive GPS coordinates through DST data and OGCMs. The current approach is to develop a more comprehensive framework that can be used for scientific research and educational purposes. The framework should build on the approach from the earlier solution, by including new parameters to generate trajectories that consists of GPS coordinates that accurately recreate where fish have been. The framework should also visualise the trajectories in order for experts to analyse them.

Data defining trajectories are GPS coordinates of locations between the release and recapture points. The challenges are how trajectories can be visualised and analysed. Visual analytics is a solution that require cross-disciplinary communities to work together to create user-friendly tools [7]. Creating a visual analytics tool for the data problem requires combining expert knowledge from movement researchers, ecologists, biologists, mathematicians and system developers. Understanding movement of species is important in order to validate trajectories [8]. Ecologists provide knowledge on fish ecosystems [9], biologists determine fish species and spawning patterns, mathematicians can create algorithms to approximate fish movement, and system developers create the application.

From a collection of trajectories, a visual analytics tool should be able to compare trajectories and determine a representative Trajectory for ensemble of derived trajectories. Calculating the representative from the collection of trajectories would result in an averaged trajectory that have characteristics that none of the trajectories from the collection have. This average may lack important information on fish migration patterns, since e.g., an averaged trajectory might choose locations that a fish would not choose due to physical barriers to movement, e.g. ocean current [10].

## 1.3 Available Data

The IMR provide necessary data to create a solution that generate fish trajectories. The extracted variables are shown in table 1.1. The provided DSTs are text files containing temperature and depth values in time intervals of 10 minutes from the release location to the recapture location. The provided OGCMs are complex three-dimensional (3D) models that store ocean variable values, e.g. temperature, depth, ocean current, to observe and monitor changes in the ocean.

| | |
|---|---|
| **Digital storage tags** | |
| Variables | Temperature |
| | Depth |
| Time Interval | 10 Minutes |
| Available Tags | DST 742 |
| | DST 1664 |
| **Ocean General Circulation Models** | |
| Variables | Temperature |
| | Depth |
| | $\eta$ (x-coordinate) |
| | $\xi$ (y-coordinate) |
| | $\sigma$ (z-coordinate) |
| | Latitude |
| | Longitude |
| | Ocean Current |
| Time Interval | Daily |
| Available Models | Nordic Seas, Incl. North Sea |
| | Norwegian Sea |

Table 1.1: Available data for simulating trajectories

## 1.4 Goal

The main goal of this project is to create a framework that simulate and visualise fish trajectories on a two-dimensional (2D) map. Trajectories are generated through linking simulations that use temperature and depth data from DSTs to GPS locations derived from OGCMs. A second goal is to have a system that runs in real time, which means that the user can alter parameters and instantly see the results on a 2D map. A third goal is that the resulting framework can be used by experts to determine ecological characteristics from derived trajectories. This means to understand where fish are spawning, feeding and their migration patterns.

## 1.5 Related Work

The paper "The dispersal pattern and behaviour of Atlantic cod (Gadus morhua) in the northern Gulf of St. Lawrence: results from tagging experiments"[11] focus on the release and recapture locations of Atlantic cod from 1995 to 2008. The goal of the paper was to use data from DSTs to analyse the dispersal pattern of Atlantic cod that inhabit the Northern gulf of St. Lawrence on the east coast of Canada. By looking at the release and recapture locations from the DSTs, changes in the migration and knowledge on their dispersal patterns were discovered by including factors, e.g. temperature.

Another paper, "Consistency in the behaviour types of the Atlantic cod: repeatability, timing of migration and geo-location"[12], use DSTs and tidal models to geo-locate Atlantic cod in the waters around Iceland. From this data, they managed to distinguish between frontal and coastal cod based on behaviour, as well as geo-locate locations of cod. The time span between each location was four months.

The paper: "Introducing a method for extracting horizontal migration patterns from data storage tags"[6] describes how the horizontal locations that forms a trajectory are found.

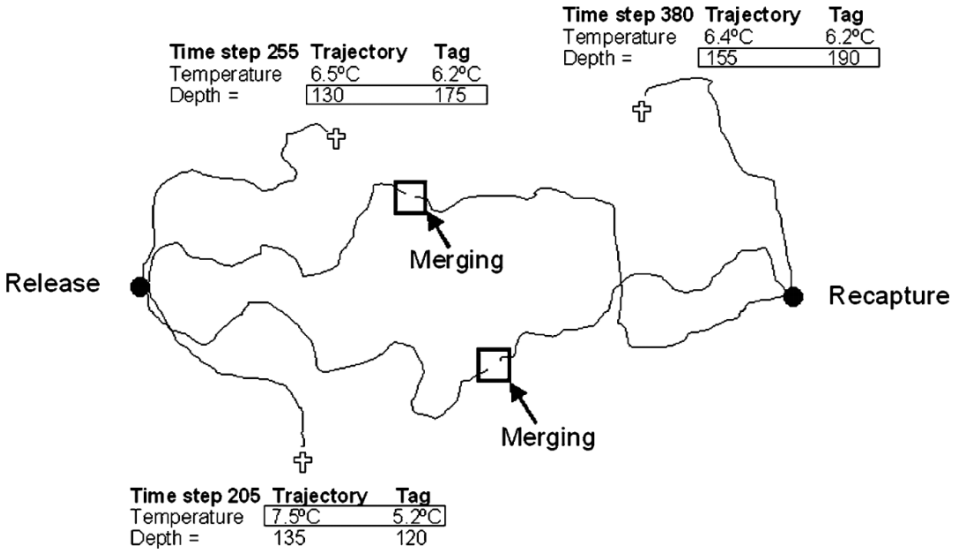



Figure 1.2: Illustration of the merge algorithm [6]

The approach in the paper is to start a number of trajectories that utilise a biased random walk algorithm. This is done in order to generate GPS locations that move from release location towards recapture location. Trajectories that find locations in OGCMs that do not match the data from DSTs will be removed. This means locations with DST depth lower than seabed or locations with temperature not within an margin of error. Originally, trajectories were started in the release location and traversed towards the recapture location using the biased random walk algorithm. Within the algorithm there is a deterministic velocity that pulls new locations towards the recapture location, and a random velocity. In the beginning of the trajectory, the random velocity would dominate when selecting next the locations. However, the deterministic component would become more dominating as the trajectory locations approached the recapture location. This led to trajectories traversing increasingly in a straight line as the locations approached the recapture location. For this reason, the application starts trajectories in both the release and the recapture location and then move trajectories towards each other. When halfway through DST data, trajectories within 5 kilometres (km) of each other are combined into one trajectory, see figure 1.2.

## 1.6 Research Question

This thesis consists of two individual applications combined into one system. The first application will simulate trajectories with GPS locations using data from DSTs and OGCMs. This application will be based on the approach from paper [6], presented in chapter 1.5. The main challenge is that the only GPS locations known are

the coordinates for the release and recapture locations. Data from DSTs only contain temperature and depth observations in time intervals. GPS coordinates must be approximated for each location between the release and recapture location by combining the DSTs and OGCMs. This results in the first research question:

**Q1. What parameters can be added to a solution that uses temperature and depth observations from DSTs and OGCMs to generate trajectories that are more realistic than the earlier solution?**

The sense of vision and the ability to visualise is essential for the brain to process new information. It assists in the comprehension and portrayal of massive amounts of data, it may lead to patterns present in the data to emerge that would otherwise remain hidden, and corruption in the data can easily be detected [13]. The second application will visualise trajectories generated by the first application on a 2D map. The main challenge is to visualise trajectories in a way that provide experts with knowledge on migration patterns and fish behaviour. This would require adding functionality that lets the user analyse trajectories and their information. This results in the second research question:

**Q2. How should generated trajectories be visualised for scientists to find and analyse their ecological characteristics?**

A framework for generating potential trajectories would have to handle big amounts of data. OGCMs consist of several gigabytes of data for one day. Creating a framework that can deliver trajectories in real time would require a way to process data fast, but without exceeding the system memory. This results in the third research question:

**Q3. How can generating trajectories be optimised to reduce the run time compared to the previous solution?**

To create a framework with analytic tools, investigating what functionality that can provide experts with knowledge on migration patterns and determine ecological characteristics is important. This results in the last research question:

**Q4. What functionality can be implemented into the framework to provide scientists with information on migration patterns and ecological characteristics?**

## 1.7 Research Method

In order to answer the research questions the thesis will mainly require a qualitative methodology. The research will be conducted on experts, chosen by the external supervisor. Since the office is located at the IMR, the experts are located close by. This makes it easier to conduct interviews and to get answers on questions related to their education. The external supervisor will be available through meetings and emails, answering questions and putting us in contact with other experts.

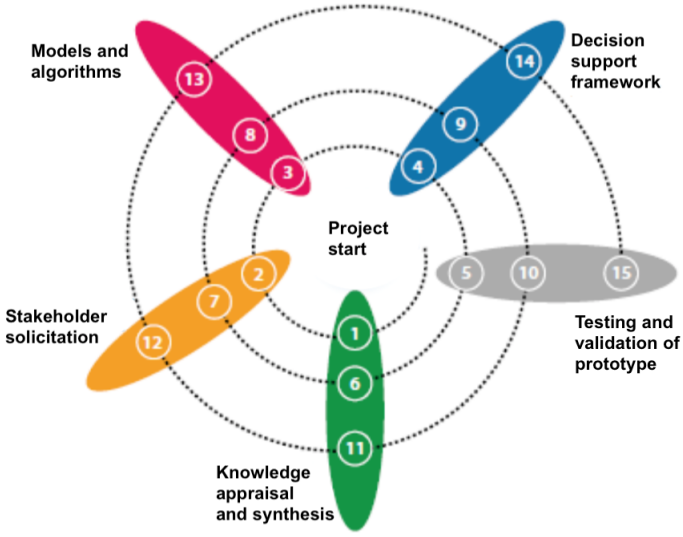



Figure 1.3: The spiral development model (SDM) for the framework

The framework requires feedback on the usability. The feedback will be provided by the selection of experts. In order to collect verbal and written data, the work in this thesis will follow a spiral development model [14] in Figure 1.3. Spiral development is a family of software development processes characterized by repeatedly iterating a set of elemental development processes and managing risk so it is actively being reduced. The framework uses the SDM in designing and stage-wise prototyping to optimize the process of knowledge discovery and integration, and impact appraisal.

The first phase, knowledge appraisal and synthesis, gather knowledge on what to create and what fields to study, e.g. behaviour of cod. The second phase, stakeholder solicitation, is to talk to experts and get their opinion on what fields that needs more knowledge, in order to understand what to create. The third phase, models and algorithms, is to create algorithms based on the knowledge learned from the previous

phases that solve challenges, e.g. how to create trajectories from data within DSTs and OGCMs. The fourth phase, decision support framework, is to create prototypes of the framework. The fifth phase, testing and validation of framework, is to test the framework on experts in order to collect data regarding improvements and functionality to be implemented in the next iterative process (spiral model).

The spiral model ensures that the decision support framework is adaptive to applying new functionality. It also eases the implementation and maintenance of existing functionality in the framework, and allows for interactive involvement of the experts, which is essential for the development of the system.

In order to iterate over each phase three times, 15 steps in the SDM is chosen. This results in more than one prototype, which makes it easier to match expectations of experts.

# Chapter 2

# Background

## 2.1 Data and Models

### 2.1.1 Tag Data

The IMR carry out systematic annual tagging programs on fish to improve knowledge on migration patterns and spawning grounds. Using various types of fish tags, they gather information that can potentially tell them if fish migrate to same fjords to spawn, time of spawning, temperature preferences, and more.

**Conventional** tags only contain an ID number and an address, which means that they do not store any information. Using an unique ID number provide researchers with knowledge on where the fish was released and where it was recaptured. Through algorithms, the researchers can approximate the age of the fish as well as how many times it has spawned.

**Electronic** tags, known as DSTs, store information in time intervals and contain sensors that register depth, temperature, salinity and light intensity. Electronic tags do not register GPS locations of fish, because this would increase the size of the tag and the antenna must regularly be above sea level to transmit GPS signals [15].

**Acoustic** tags transmit sound signals every 1.5 second. The sound signals have a strength of 158 decibels that are picked up by three positioning buoys on the surface. When transmitting sound signals, each of the three positioning buoys must receive the signals in order to accurately define the GPS location and depth of the fish. The three positioning buoys receive sound signals from multiple fish at the same time. In order to separate between the fish that transmit sound signals, each tag transmit a unique sound signal that is at a different frequency than the other tags [16].

**Satellite** tags measure temperature, depth and light intensity every two minutes. Satellite tags can remain attached to the fish for up to a year. At a given time, a mechanism releases the tag from the fish and the tag floats up to the surface and transmit the minimum and maximum values to a satellite. The data is then further transmitted from the satellite to the IMR. In order to access all the recorded data, the tag has to be found and returned to the IMR [17].

**Passive Integrated Transformer (PIT)** tags are internal tags that do not actively register data. They function as an electric coil that transmit the id number of a fish, when it passes the magnetic field of an antenna. If the IMR set up an antenna, they can see how many times a fish encounters it. This can determine how often fish are in specific areas [18].

The data provided by the IMR for this thesis are data within electronic tags, DSTs. They consist of depth and temperature values every 10 minutes, from the time the fish is released until it is recaptured.

### 2.1.2 Ocean General Circulation Models

Ocean general circulation models (OGCMs) are three-dimensional (3D) models that describe the climate in the ocean through measuring ocean variables, e.g. temperature, depth, ocean current. The models consist of GPS coordinates and a 3D grid. The grid is built up by horizontal coordinates where $\eta$ (x-axis) is the direction from east to west and $\xi$ (y-axis) is the direction from north to south, and vertical coordinates where $\sigma$ (z-axis) is the direction from sea surface to seabed.

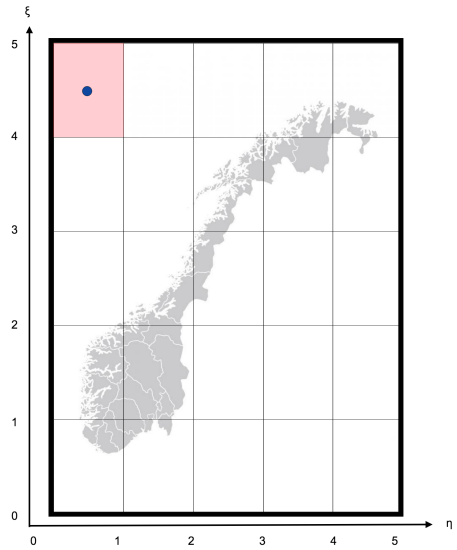



Figure 2.1: Painted grid point where ocean variable values are measured in the centre (blue circle)

In the horizontal, the resolution of a model describes how accurate the data set represent ocean variable values. The resolution is the size of a grid point, where the geographical area within a grid point contains the variable values located in the centre of the grid point. Figure 2.1 illustrates a painted geographical area that contain the variable values from the blue circle in the middle of the grid point. This means that all GPS coordinates within a grid point returns the same ocean variable values.

The resolution is determined by the sizes of the x and y-axes, where small sizes of x and y result in large grid points (low resolution), and vice versa. Figure 2.2 illustrate two models that cover the same geographical area, but with different sizes on the x and y-axes. The figure on the right has four times the amount of grid points than the figure on the left, and measures four times the amount of ocean variable values. To represent ocean variables as accurate as possible, it is important to have high resolution models (small grid points) to keep the geographical area the variables cover small.

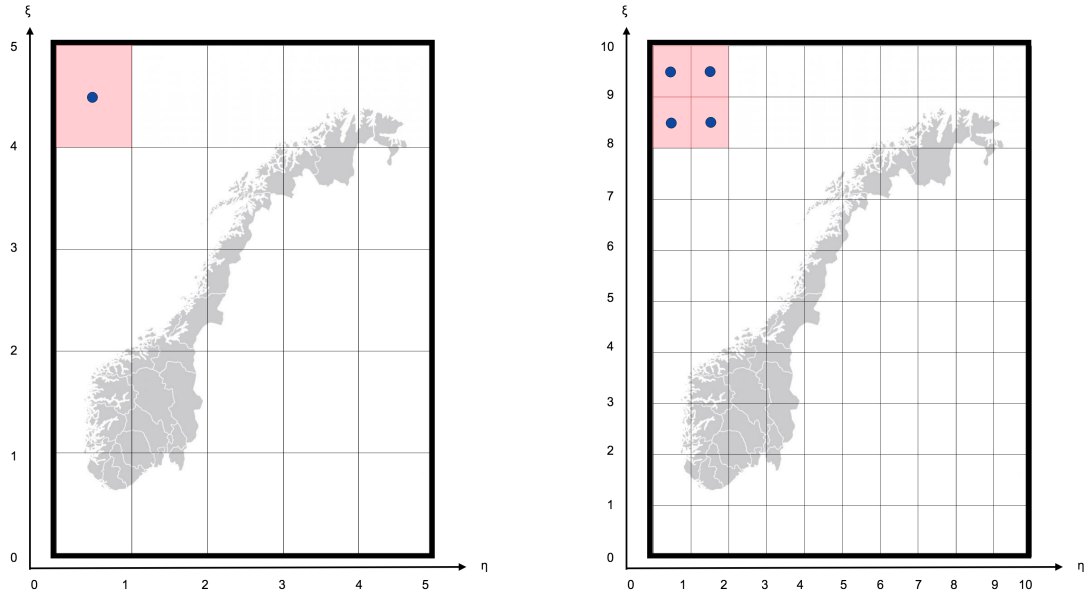



Figure 2.2: Horizontal representations with different sizes on $\eta$ and $\xi$

The vertical z-axis exist in a (x,y)-point if the depth at point (x,y) is larger than ten meters. Grid points with depth values less than ten meters are likely to be close to land and have a high probability for partially crossing land in the grid point. These points are registered as grid points on land, and they do not contain the vertical z-axis. Grid points with vertical z-axis always have the same number of vertical layers regardless of depth [19]. This is because the vertical layers use the $\sigma$-coordinate model [20] shown in Figure 2.3. This model follows the underwater terrain; where terrain is sloped, so are the vertical layers. This ensures that there is always the same number of vertical layers regardless of depth.

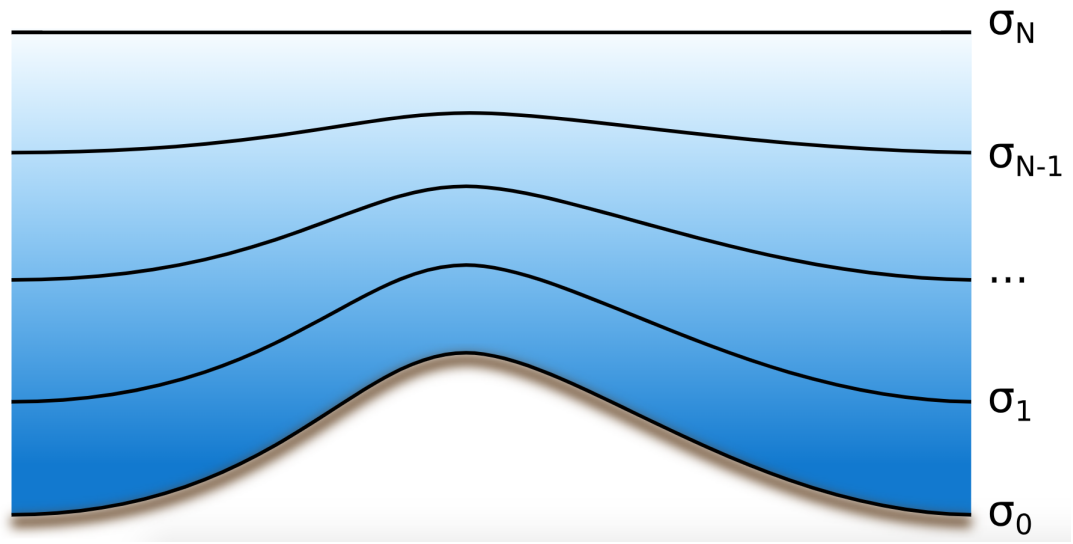



Figure 2.3: Schematic of a $\sigma$-coordinate model [21]

Each grid point has ocean variable values, e.g. temperature and depth, measured in the centre of the horizontal point at each vertical layer. OGCMs can measure ocean variables at hourly, daily or monthly time intervals [22].

The IMR have OGCMs for the Nordic Seas including the North Sea, as well as models for the Norwegian Sea. The Nordic Seas model is shown in Figure 2.4, where the painted area displays the geographic area the grid covers. This model has an x and y grid size of 580x1202 and to cover the geographic area with this grid size, the resulting grid has a resolution of 4x4 km.

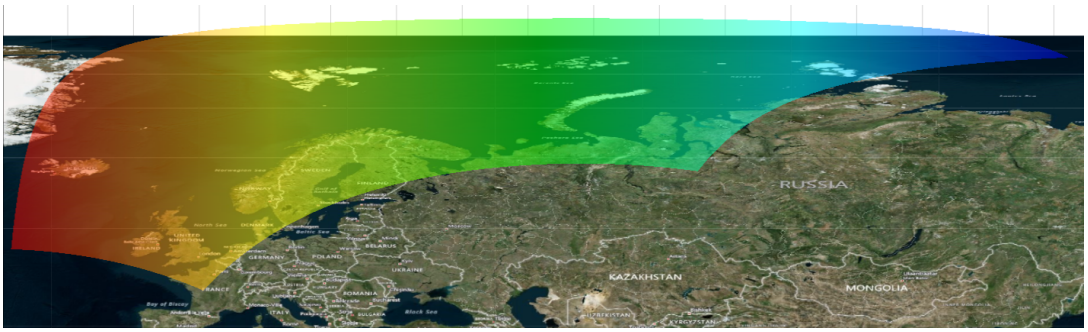



Figure 2.4: Ocean general circulation model for the Nordic Seas, including the North Sea

The model for the Norwegian Sea is shown in Figure 2.5, where the painted area displays the geographic area the grid covers. This model has a x and y grid size of 902x2602. Compared to the model for the Nordic Seas, the Norwegian Sea model covers a smaller geographical area, but the grid size is larger. This results in a grid with resolution of 800x800 meters, which is higher than the resolution in the Nordic Seas model. The ocean variable values in the Norwegian Sea model, represent ocean variables more accurate than the values from the Nordic Seas model because each grid point is 80% smaller than grid points in the Nordic Seas model. This means that one grid point in the Nordic Seas model is represented as five grid points in the Norwegian Seas model.

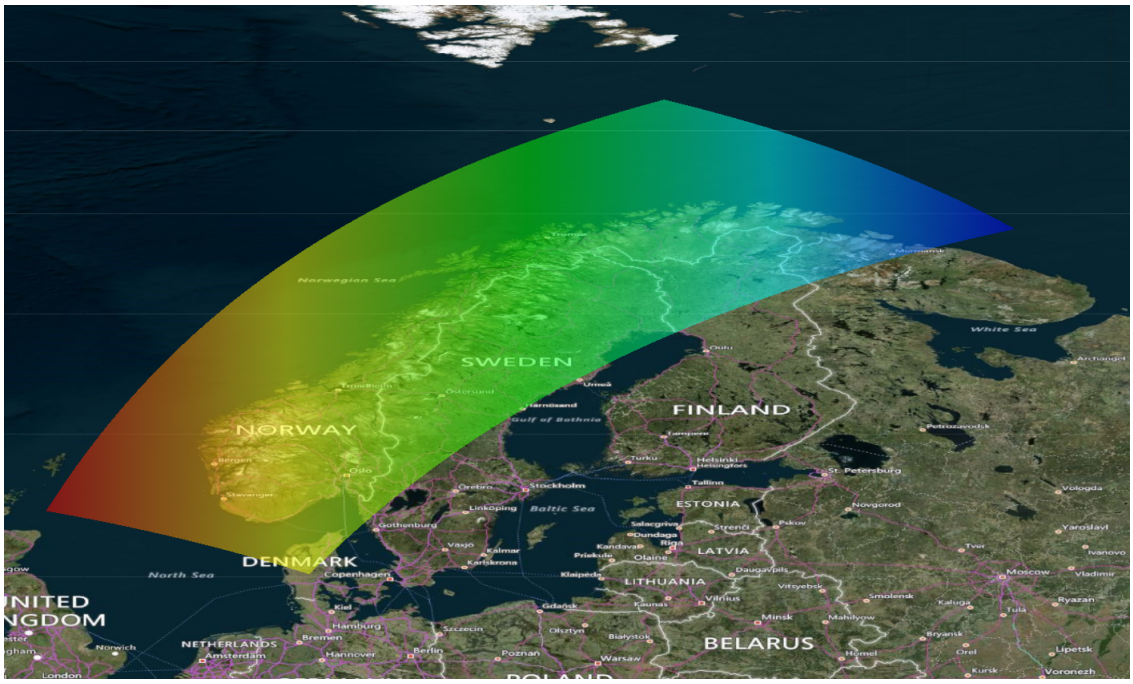



Figure 2.5: Ocean general circulation model for the Norwegian Sea

### 2.1.3 NetCDF

NetCDF is a set of libraries that can be implemented in a program to allow it to read and write NetCDF files. These files contain array-oriented scientific data [23], such as the OGCMs explained in section 2.1.2. The data can be accessed through a simple interface, while array values can be accessed directly without knowing how data is stored. This means that the values can be fetched with the correct parameters. Tools and application programs can access NetCDF data sets and transform, combine, analyse or display specified fields of data [24].

## 2.2 Simulation Application

To create the simulation application that generate trajectories, a programming language must be selected. This section will go through potential programming languages and conclude with the one that will be used in this project.

### 2.2.1 .NET Framework

.NET is a framework for building and running applications. The core features of .NET are the Framework Class Library (FCL) and Common Language Runtime (CLR). FCL is a collection of reusable classes, interfaces and value types. CLR manages the execution of .NET applications and converts compiled code into machine instructions for the central processing unit (CPU) regardless of what programming language it is written in [25]. This means that different programming languages can communicate with each other in the same system, which is advantageous since different programming languages are optimised for specific tasks [26].

.NET allow developers to share code through NuGet packages. The NuGet packages contain compiled code that expands the library of the application [27]. Scientific DataSet Lite (SDSLite) 1.4.0 is a NuGet package that can read and write matrices and multidimensional grids which are common in scientific modelling [28]. This package makes it possible to manipulate NetCDF files from an application written in C#.

C# is therefore a programming language that can be used to create the algorithm that simulates possible trajectories for fish. An additional argument is that technologies for visualising the algorithm supports applications in C# [29].

### 2.2.2 Python

Python is a high-level programming language created for general programming. The design focuses on code readability achieved by using an abundance of whitespace [30].

It manages the memory automatically and has a dynamic type system which means that variables are not bound to a type. In addition, there are several programming paradigms that are supported, such as object-oriented, imperative, functional and procedural [31].

Python offers several functions for handling NetCDF files and are therefore a viable alternative for the simulation application. It can implement two libraries that would be relevant, they are NetCDF4 [32] and NumPy [33]. NetCDF4 is an interface to the NetCDF C library that allows the program to read and write to a given NetCDF file. NumPy offers scientific computing as well as a powerful multidimensional container for generic data. The container would be required to store data from the NetCDF files that are read using the NetCDF4 library.

### 2.2.3 Choosing .NET Framework

The .NET Framework was chosen for this project because C# is a statically typed language [34]. Meaning that the type of a variable is known when it is compiled. This reduce the number of minor bugs because the compiler catches them early on. This is not the case with Python since it is a dynamically typed language [34], it will throw an existing exception at run time. C# also has a speed advantage over Python because it is compiled [35].

## 2.3 Visualisation Application

To create the application that visualise trajectories on a 2D map, a technology must be selected. This section will go through potential technologies and conclude with the one used in this project.

### 2.3.1 Game Engines

Game engines are a software development environment that has been made for the purpose of developing video games [35]. It abstracts tasks common to games like rendering and physics. This is to make it easier for developers to focus on important features for their game instead of using time on physics, movement and light sources [36].

### 2.3.2 Visualising Simulated Trajectories

Simulated trajectories can be visualised using a game engine, which would allow for more optimisation and control over the representation. This approach will require that the user has a powerful system, otherwise the software could be strained for the resources it would need.

It is possible to visualise simulated trajectories in a web browser using a WebGL API [37]. This would allow for easy sharing of the solution to anyone who want to use it if they have the correct Uniform Resource Locator (URL). For visualising trajectories on a 2D map, this should not be a problem. However, the problem is that simulating fish trajectories will require a lot of computational power.

### 2.3.3 Unity3D

Unity is a game engine that can be used to create games in 2D and 3D. C# is used as the primary scripting language, but also includes UnityScript [38]. There are 27 supported platforms allowing developers to publish their game to whatever platform they want [39].

It provides standard assets for anyone to use free of charge. Letting developers start creating their game without having to worry about designing characters and other objects for the game [40].

### 2.3.4 Mapbox API

Mapbox provides custom online maps for websites and other applications through their API. This API provides a map of the world that the user can zoom in and out on as well as move around using a mouse or the key arrows. It can be implemented in both WebGL and Unity3D [41].

### 2.3.5 Choosing Unity3D

Unity3D was chosen for this project because it can run the simulation application. Unity3D also supports exporting the software as a WebGL so the finished framework can be accessed online, but this would require uploading the simulation application to a server. Mapbox API can be used in Unity3D and provides maps of the earth that trajectories can be visualised on.

# Chapter 3

# Design & Solution

## 3.1 Application Design

The top level design of the framework is shown in figure 3.1. The user interacts with the Unity3D application that presents tools for modifying trajectories and running the simulation application. The goal of the simulation application is to find fish trajectories based on parameters set by the user.

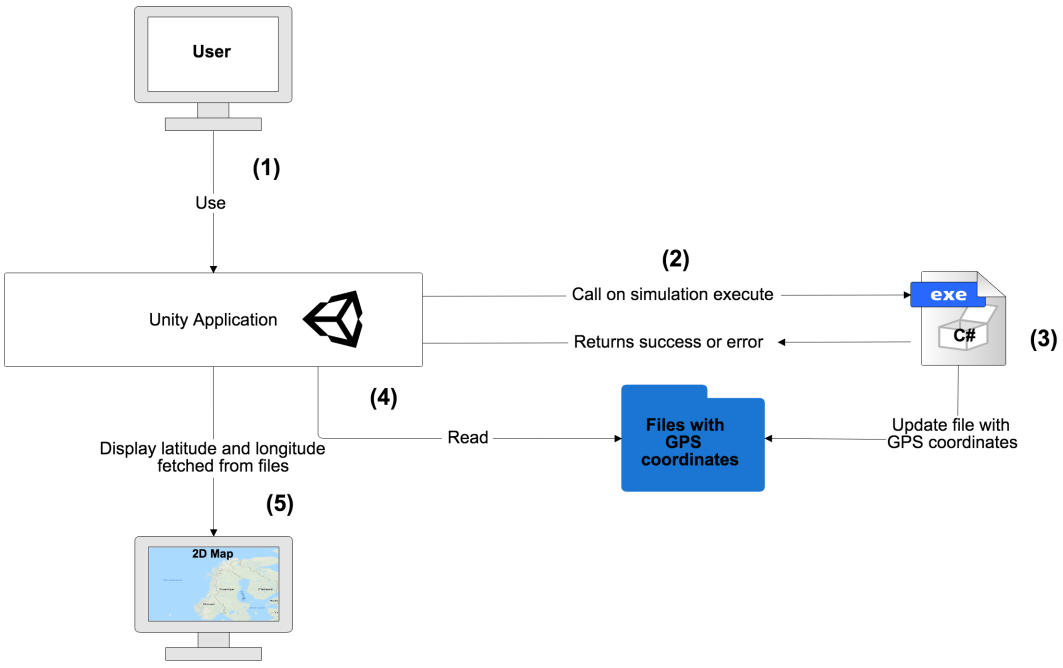



Figure 3.1: Top level design of the framework

The Unity application calls on the executable file (.exe) of the simulation application that either find or not find trajectory results. Fish trajectories are found if the simulation application manages to go through all DST data and find locations at

each step that match parameters set by the user. Each of the found trajectories are stored as a file, containing GPS coordinates of all locations in the trajectory, and placed in a file directory. Based on the result from the simulation application, the Unity application will either read the file directory and display the trajectory files on a 2D map or specify that the application did not find any trajectory results with the given parameters.

### 3.1.1 Simulation Design

The design of the simulation application is shown in figure 3.2. In step one, the simulation receives parameters set by the user such as; which DST to simulate, number of simulations, time step (see section 3.2.13), and allowed margin of error on temperature. The next step is to read a text file containing the available DSTs as well as the release and recapture locations of each fish. Then, the simulation will load several NetCDF files containing static OGCM data (see section 3.2.11) needed for finding GPS locations throughout the simulation.

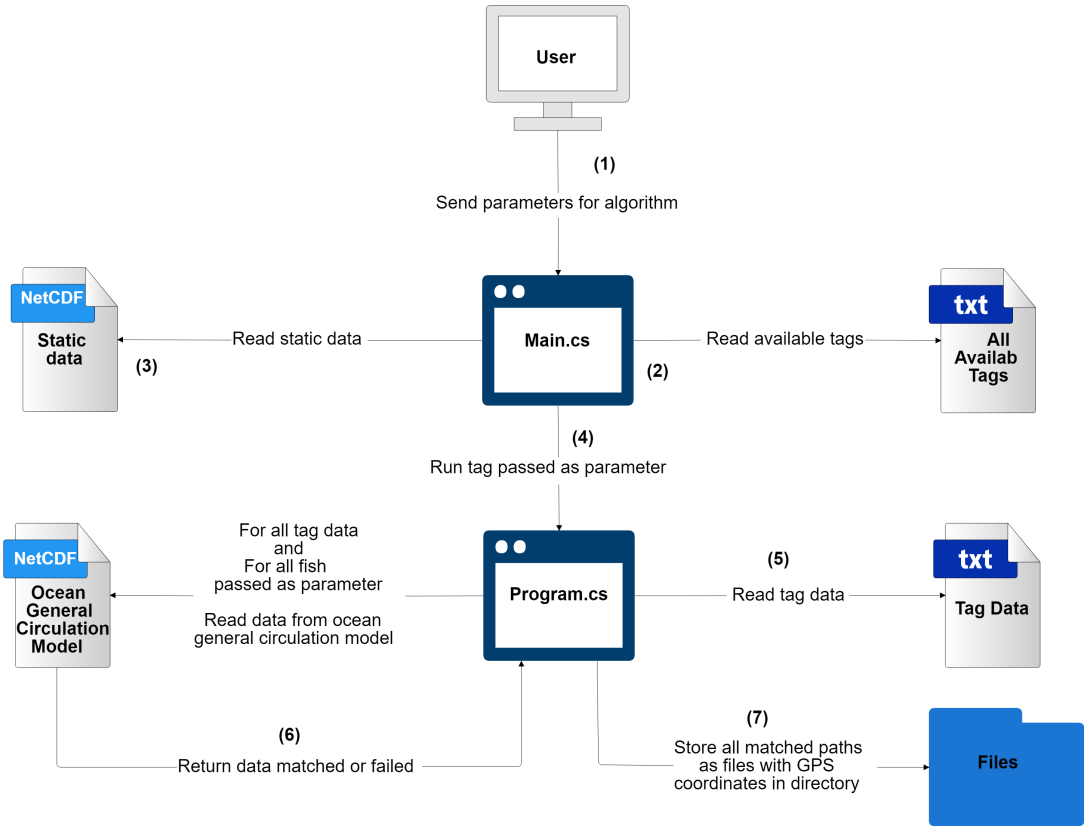



Figure 3.2: Design of the simulation algorithm

In step four, all the parameters received from the user will be passed to the controller class which will perform one of the implemented algorithms for generating fish trajectories. Either the General algorithm (see section 3.2.2) or the Merge algorithm (see section 3.2.4). The fifth step is to read all the data entries from the selected

DST from a text file. It contains depth in meters, temperature degrees in Celsius (°C), and date and time of when the data was recorded. When this is done it moves on to step six which will continue to loop until it has iterated through all the entries in the DST data or there are no more simulations being run. Every iteration of the loop starts with reading the date from the current entry of the DST data which is then used to load the OGCM with ocean variables from the same date onto the random-access memory (RAM). If it is the first iteration, the simulation will convert the release location from latitude and longitude to x- and y-coordinates. Then it will use the x- and y-coordinates to search for valid locations in the OGCM. Once the search for valid locations is complete, a number of simulations is started in order to find trajectories. Each simulation will select one of the valid locations randomly and add it to their trajectory. If no valid locations are found in the first iteration, no simulations will be started, and the simulation application will be terminated.

All of the simulations started in the first iteration will use the most recent (x,y) grid location from its trajectory to try and find new valid locations based on the data for the next entry in the DST. Then, any simulation that can generate one or more new valid locations chooses a random one and adds it to its trajectory. If none are found, the simulation is terminated. Once all the iterations are complete, the simulation application starts step seven if there are one or more simulations that were able to complete their trajectory. All the trajectories from the completed simulations are saved to individual text files, containing latitude, longitude, temperature and depth from DST, and temperature and bottom depth from OGCMs, for each location.

### 3.1.2 Visualisation Design

The design of the user interface from the visualisation application is shown in Figure 3.3. When the application is started, it reads a text file that consists of all DSTs available for running the simulation application. Without this file the application will not allow the user to run the simulation application, since it means that either DST data or OGCMs are missing. The application will then read a text file containing the parameters used in the previous execution of the simulation algorithm. If this text file does not exist, all parameters are set to zero.

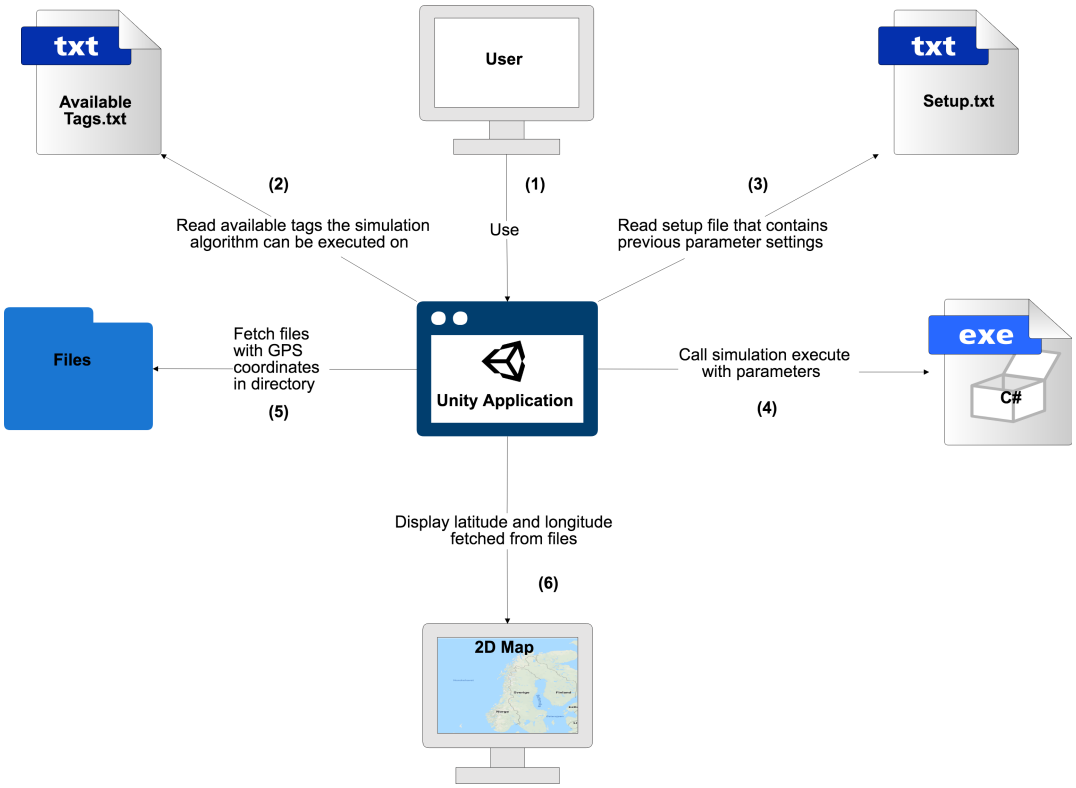



Figure 3.3: Design of the visualisation application

After the user has set new parameters, the application will run the simulation algorithm and wait for its completion. When completed, the application will fetch trajectory files from a file directory and display them on a 2D map.

## 3.2 Simulation Application

The simulation application of the framework consists of three algorithms that generate trajectories. When the simulation application is launched, it starts individual simulations that try to generate their own trajectory from release location to recapture location of fish. The approach of the simulation application and how the three algorithms generate trajectories are explained in the sections below.

### 3.2.1 Simulation Overview

The goal of the simulation application is to run a number of simulations in order to generate potential trajectories for a fish by using its DST data and OGCMs. Due to large amounts of data, the data used needs to be limited. This is done in order to keep the trajectories from being too complex and the execution from taking too long time.

#### Time Step

A trajectory consists of GPS locations from the release location to the recapture location, where the amount of GPS locations is determined by how often DST data is read. DST data is measured every ten minutes, which results in large numbers of GPS locations with short distance between each other. In order to limit the amount of GPS locations, a parameter for determining the time step is necessary. This lets the user decide how often the algorithm should read DST data, e.g. read one DST data per day, read one DST data every second day, etc. An example of this is a fish that has measured data for one year. If the simulation algorithm should calculate GPS locations for the values measured every ten minutes, it would result in:

$$365 \text{ days} \times 24 \text{ hours} \times \frac{60 \text{ minutes}}{10\text{-minute step}} = 52\ 560 \text{ GPS locations}$$

If the simulation algorithm instead calculates GPS locations for one data value per day, it would result in:

$$365 \text{ days} \times 1\text{-day step} = 365 \text{ GPS locations}$$

The difference between the amount of GPS locations are huge, but their trajectory results would look similar. The distance a fish can reach is scaled according to how often DST data is read. Instead of keeping track of every little movement, a larger movement would result in similar trajectories.

#### Finding GPS Locations

The first GPS location in a fish trajectory is the release location of the fish. This location is converted into x- and y- coordinates in an OGCM. Depending on the

time step, the maximum distance the fish can reach in each step is calculated. This distance is used to find (x,y) grid points that are within range of the next location in the DST data. Figure 3.4 display the current location (blue circle) and all grid points (red) calculated within range of the next location. If the maximum distance is larger than illustrated in Figure 3.4, the surrounding grid points would be further away from the current grid point, as the fish would have potential of reaching a greater distance.

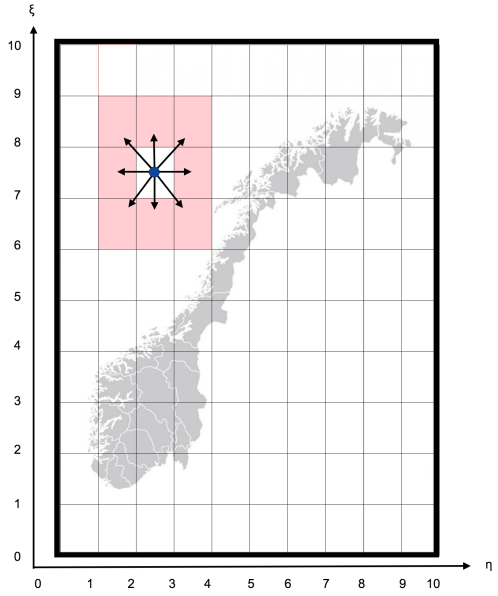



Figure 3.4: Grid points (red) within reach of current grid point (blue circle)

Each of the (x,y) grid points within the range of the current location are possible locations the fish could traverse to. In order to eliminate possible locations that do not fit the recorded DST data, depth and temperature are used. To keep a possible location, the depth in the grid point must be greater than the depth in the DST data, and the difference between the temperature from the DST data and the grid point must be within an margin of error set by the user, see section 3.2.8. The next location in the trajectory is chosen from the remaining possible locations. This routine is repeated for all the selected DST data in the time step. When one of the possible locations is chosen as the next location in the trajectory, latitude and longitude within the (x,y) grid point are stored.

### 3.2.2 General Algorithm

The main solution for generating trajectories is to run several simulations from the release location of a DST and weighting each simulation towards the direction of the recapture location. The weighting makes it more likely that the next location within a simulation is closer to where the fish was recaptured. Figure 3.5 displays

three simulations with different scenarios. The simulations went through sixteen time steps, which results in sixteen locations in a completed trajectory.

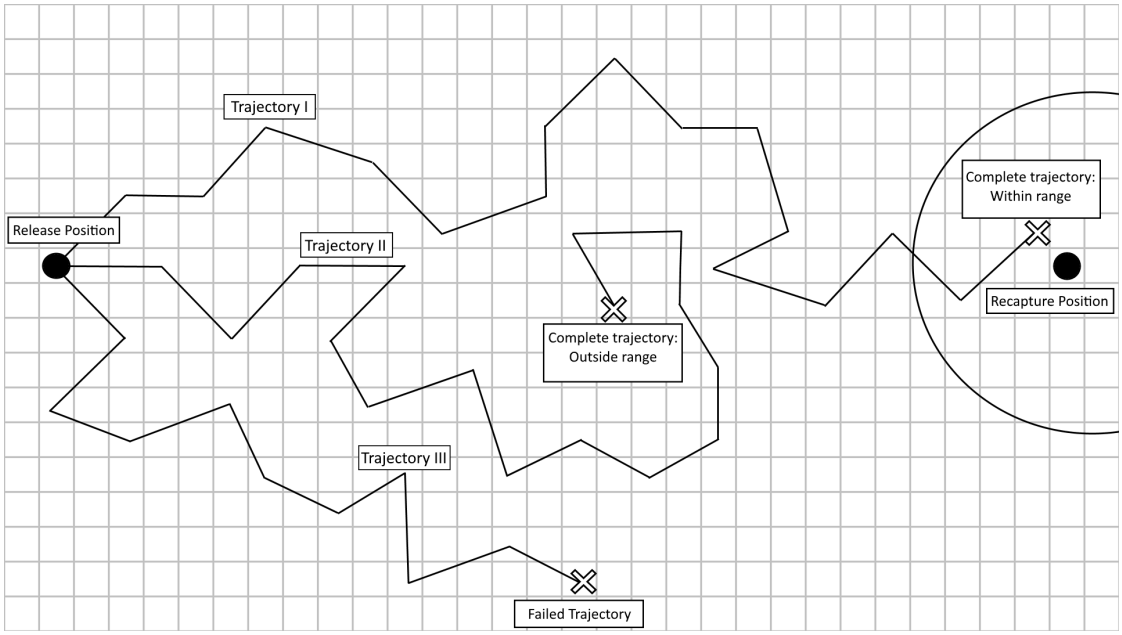



Figure 3.5: Illustration of the General algorithm for three trajectory scenarios

Trajectory I) is a plausible trajectory since it has sixteen locations, where the last location is within range of the recapture location. Trajectory II) also has sixteen locations, but is less plausible since it is outside the range of the recapture location. The range is equal to the maximum distance a fish can travel in one time step (see section 3.2.10). Even if a trajectory is not within range of the recapture location, it will still be stored in a separate file directory and shown to the user. The reason for this is that the recapture location can be wrong due to fishermen catching tagged fish at sea, but do not register the recapture coordinates until they reach the harbour. Trajectory III) is a failed trajectory since the simulation did not find a location at each step. This happens when a simulation travels in a direction that does not have any locations that satisfies the temperature and depth requirements.

### 3.2.3 Release Continuously Algorithm

Rather than starting all the simulations in the release location of the DST like the General algorithm does, simulations are only started when a given simulation finds more than one new valid location. In each simulation, the current location is used to find valid locations for the next time step. If no valid locations are found the given simulation is terminated, if one valid location is found the simulation chooses the valid location as its next location in the trajectory. However, if the number of valid locations are between $2, ..., n$, new simulations are started. If the given simulation chooses one of the valid locations, $n - 1$ valid locations will remain unvisited. In

order to visit $n$ locations, the given simulation must be duplicated $n$ times. This means that there exist $n$ simulations with the same previous locations as the given simulation. Each simulation chooses one of the valid locations as the next location in their trajectory, resulting in all valid locations being visited.

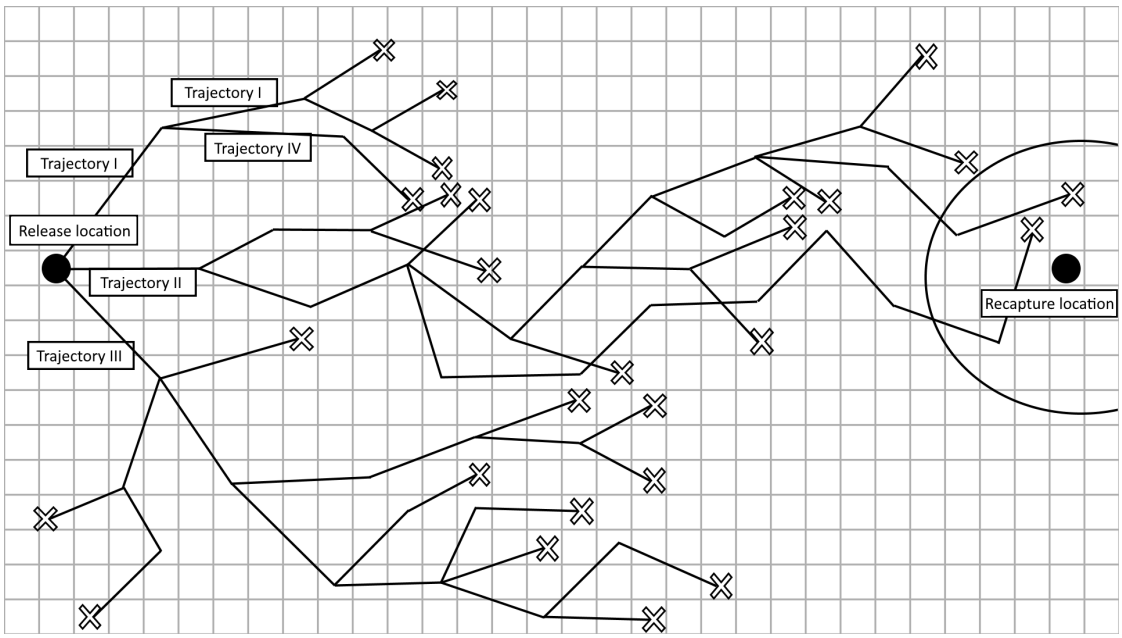



Figure 3.6: Illustration of how the Release Continuously algorithm works.

In Figure 3.6 the simulation application finds three valid locations within range of the release location. This results in three simulations being started.

The general algorithm would have all the simulations randomly choose one of the new locations and add it to its trajectory even if there are 10 000 simulations and only three locations. However, this algorithm would instead start only three simulations and each simulation will choose one of the new locations that the others did not. An example of this is shown in figure3.6 with three trajectories. This will save the user a lot of time because there will be less calculations for the computer to process. The second iteration of trajectory I) in figure 3.6 finds two new valid locations. It will then randomly choose one of the new locations and add it to its trajectory. Then a new simulation called Trajectory IV is created with the same trajectory locations minus the location that Trajectory I just added. It then adds the remaining location to its trajectory. The simulation application will continue to do this until it has fully completed every iteration. Then it will validate the trajectories of all the completed simulations the same way that the general algorithm does by checking if the last location of the trajectory is within range of the recapture location. This was done to cut down the run time of the simulation so that the user could either run it with more simulations or spend less time waiting.

There are few new and valid locations in the beginning which means there are few simulations and therefore the simulation application starts off quick. However, over time it starts to slow down because for each iteration that the simulation application goes through the number of simulations increases. If this simulation application was left to run with no limitations it would take an extraordinary long time to complete. Therefore, the user must set a maximum number of simulations that the algorithm can start. Once this maximum is hit the algorithm will no longer start any new simulations. With this limitation it will still slow down over time but will speed up again after the maximum number of simulations has been reached. This is because the number of simulations determines how many calculations must be performed. Therefore, as the number of simulations increases so will the number of calculations, which takes more time. On the other hand, when the number of simulations is reduced there are fewer calculations which means it will go through the remaining steps faster.

Unfortunately, this algorithm has problems with producing trajectories. Since this algorithm starts simulations for all valid locations, some of those may be in directions that would normally be prevented by the weighting. Furthermore, this can cause the simulations to travel too far from the recapture location while starting new simulations that will also travel too far from the recapture location. Because of the high temperature halfway through the DST 742 most of the trajectories are terminated.

### 3.2.4 Merge Algorithm

The idea is the same as the General algorithm, but it is run twice and from each end for half the data in the DST. Meaning that half of the simulations are started in the release location and moved towards the recapture location, but they go through only half of the data in DST. These simulations generate the forward trajectories. The other half of the simulations start in the recapture location and move towards the release location by starting with the last entry in the DST and then iterating backwards through the data. This generates the backwards trajectories. When the simulations from both ends are finished the simulation application goes through all the trajectories that were completed and the once that are close enough to each other are combined. The distance between two locations must be less or equal to the maximum distance that a fish can traverse in one time step for two trajectories to be combined into one complete trajectory.

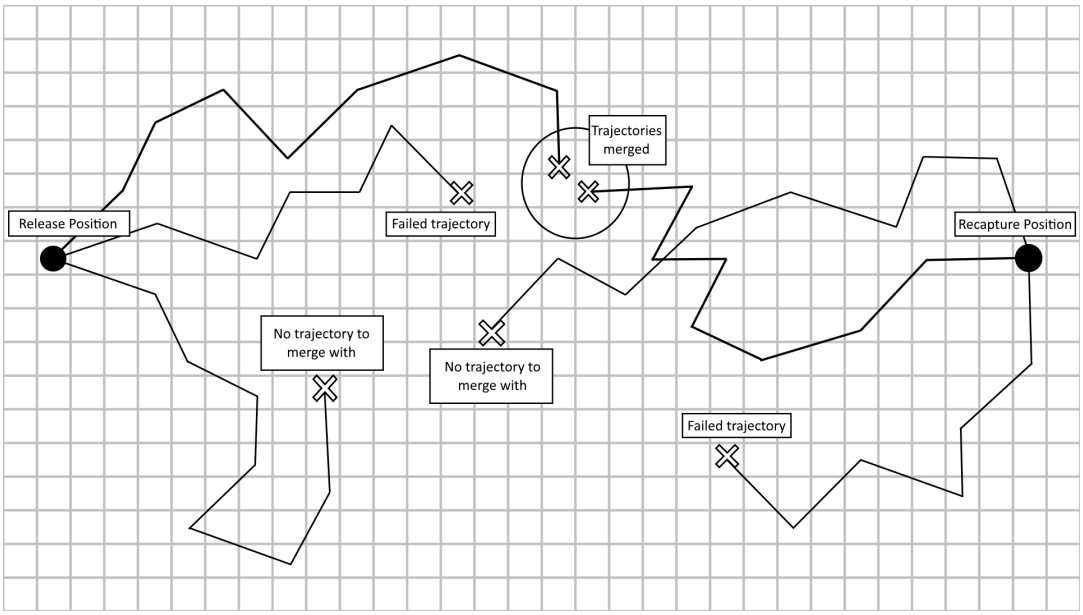



Figure 3.7: Illustration of how the Merge algorithm works with the three possible outcomes for a trajectory.

An example of this can be seen in figure 3.7 which illustrates the three scenarios that can occur during a simulation. In this example a total of six simulations are run. Three simulations are run from the release location and weighted towards recapture location, and three simulations are run from the recapture location and weighted towards the release location. The first scenario is two trajectories from completed simulations that are combined because both have a location for every iteration and are close enough to each other to be combined into one trajectory. The distance between the two locations must be lower than the maximum distance that a simulation can cover in one iteration. Then the merged trajectory is stored in a text file. However, there are not always other trajectories that are within range so even trajectories from completed simulations will be discarded if there are no other trajectories that it can be combined with. Lastly, there are simulations that cannot find any new valid locations and that are terminated.

This algorithm was implemented for two reasons. The first reason is that the paper the thesis is based on used an approach like this [6]. The paper did it because the deterministic component in the random walk cycle would dominate towards the end and force all the simulations to travel straight to the recapture location. Therefore, having the simulations travel half the distance from each end would avoid the deterministic component becoming too dominant which is why the merge algorithm was used instead [6]. Secondly, most of the simulations are terminated about halfway through the simulation application because of a significant increase in temperature in the DST data. This increase causes most of simulations to terminate if they are not close enough to Lofoten where the temperature is higher. Therefore, having the

simulations travel only halfway from each end might lead to more of them making it through.

The resulting trajectories have several locations that are clustered around the start location for the first seventy days of the simulation. Afterwards all travel towards the recapture location with one or two clusters on the way. Once the trajectories reach the end it does not travel around much. Indicating that the fish did not spend much time in Lofoten. However, it could be that the weighting in the merge algorithm pulls the fish away too soon.

### 3.2.5 Comparison of Algorithms

Unfortunately, the release continuously algorithm is unable to generate any trajectories. The reason is unknown and there was not enough time left to fix it. Therefore, only the merge and general algorithm will be compared. In terms of the number of trajectories that are generated by the two algorithms, the merge algorithm is significantly better when testing with the DST 742. Testing has shown that with the same settings the merge algorithm, generates more than twice as many trajectories as the general algorithm. Even though the merge algorithm generates more trajectories, it also consistently takes almost twice as long to complete as the general algorithm. This can be mitigated by reducing the margin of error on temperature. This will reduce the number of trajectories that are generated by both algorithms but could give trajectories that are more realistic. An additional advantage of the merge algorithm is that it can produce trajectories with a lower margin of error on temperature than the general algorithm. The testing discussed above used a margin of error of 1.2 °C and when that was reduced to 1 °C the merge algorithm looked even better. Instead of producing twice the number of trajectories it produced five to ten times the number of trajectories. It still needed more time to complete, but rather than taking twice as long the difference was reduced to roughly 40% instead. Furthermore, reducing margin of error on temperatures down to 0.9 °C lead to the general algorithm no longer being able to generate any trajectories at all. However, the merge algorithm was still able to produce trajectories. The merge algorithm can generate trajectories with a margin of error on temperatures as low as 0.76 °C.

### 3.2.6 Ending Simulated Trajectories

Once simulations of the DST 742 eventually reach Lofoten all of them go back and forth within Lofoten. It is also difficult to see a pattern even after visualising the resulting trajectory. Furthermore, the temperature data for Lofoten is not very accurate according to experts at the IMR. Therefore, it may be pointless to have the simulation go back and forth in Lofoten until the final day is reached. So, when the

simulation is within a certain radius of the recapture location it can be considered complete. This change could lead to the simulation being able to generate additional trajectories. Also, the margin of error on temperatures could be lowered. This can only be done when running the General algorithm on DST 742, and not on DST 1664. This is because the release and recapture locations on DST 1664 are so close to each other that the simulation would be considered complete almost at once.

### 3.2.7 Grid Points Versus Latitude and Longitude

Initially this project attempted to determine where the fish could traverse by calculating several latitude and longitude coordinates. These coordinates had to be converted to x- and y-coordinates within the NetCDF4 files so that the temperature and depth for those given coordinates could be read and the viability of the locations could be evaluated. However, converting latitude and longitude to x- and y-coordinates was incredibly computing intensive, leading to the simulation application taking far too long to complete. Each iteration needed over 1 minute to complete when using this approach. Furthermore, one execution of the simulation application can contain over 700 time steps which would approach a run time of 12 hours which is unacceptable since one of the goals of this project is to generate the trajectories in real time. This is because each latitude and longitude location would have to go through over two million different x- and y-coordinates in order to find the x- and y-coordinates.

Therefore, x- and y-coordinates were used instead. The OGCMs have a grid in which all the squares are either 800x800 meters or 4x4 km. Meaning it could be used to calculate the distance from one (x,y) grid location to another. Furthermore, each set of x- and y-coordinates contain lat-$\rho$ and lon-$\rho$ which are real world latitude and longitude coordinates. This dramatically reduced the completion time for the simulation. It was no longer necessary to use actual latitude and longitude, apart from the conversion of the latitude and longitude of the release and recapture locations. For the first location of the simulation, where only latitude and longitude are known, every set of x- and y-coordinates are checked and the one with the latitude and longitude closest to the release location is chosen.

### 3.2.8 Validating Calculated Locations

When a simulation generates a new location in a trajectory, it must check whether the new location is valid or not. First it will check if there is land between the current location and the new location. If there is, it is removed as a potential location.

Afterwards the simulation will check if the new location is deep enough. This only

requires the x- and y-coordinates, which are already known. The depth data for all (x,y) grid locations for both the Nordic Seas and the Norwegian Sea models are stored in two arrays that are loaded from a NetCDF file on the computer at the start of the simulation. Once the depth is collected from the array it is compared to the depth from the DST data. If the location is deep enough it goes on to check the temperature.

To get this information it needs the x- and y-coordinates as well as the z-coordinate which is an index for a given $\sigma$ layer. The x- and y-coordinates are already known, but the correct z-coordinate must be found using the depth from the DST. This is done using a data set provided by an oceanographer employed at the IMR which contains the depth for all (x,y,z) grid points. To find the correct z-coordinate, a for loop goes through all the values within the z-coordinates in the known (x,y) grid point, and picks the one with the depth that is the closest to the actual depth from the DST. Then it is used to find the temperature of the location which is stored in an array. Once the temperature is collected it is compared to the temperature from the DST and if the difference between the two temperature values are less than a margin of error, the location is considered valid.

### 3.2.9 Weighting

Once a simulation has generated a list of valid locations, it will pick a random one from the list and then use weighting to determine whether it should be chosen. If it does not choose that location it picks another random location from the list and tries again until one is selected. There are two ways that the simulations will be weighted, by the distance from the recapture location and the ocean current. Meaning, if the randomly selected location is closer to the recapture location than the current one, it is more likely to be chosen. If it is not, then it is less likely to be chosen. This is done by first generating a random number which is between zero and the total number of possible new locations that can be chosen minus one. Then another random number is generated which is between zero and one. If the new location is closer to the recapture location than the current location the simulation checks if the random number is less than the threshold given by the user, which is also between zero and one. If the location is not closer than the current location, it will check if the random number is greater instead. As for the ocean current weighting the simulation will check a value called “extraWeight” which is either true or false. If it is true it means that the ocean current moves towards that location which means that location is more likely to be chosen. This is done by increasing or decreasing the threshold given by the user depending on whether the new location is closer to the recapture location or not.

### 3.2.10 Speed

It is important for the simulation that it has a reasonable speed for the trajectory. After running the simulation several times, it is clear that the simulation is sensitive to increases and decreases in speed. In early versions of the simulation application the speed was static across every iteration and any change to the speed would have a noticeable effect on how many trajectories that made it through. Later, this was changed to a random speed within a set interval. This is the formula used:

$$(\text{Fish Length} \times \text{Random Value} \times 3.6) \times (\text{Time Step} \times 24)$$

The formula is from the paper "The Virtual Aquarium: Simulations of Fish Swimming – M. Curatolo and L. Teresi" [42].

$$\frac{|V_{svim}|}{L} = 0.71f \tag{3.1}$$

In this formula the velocity ($|V_{svim}|$) of the fish is divided by its length ($L$) and the result is multiplied by the frequency ($f$) of the tail's movement in hertz. The frequency of the tail movement is unknown since the DSTs do not record it. Tail frequency is therefore replaced with a random value.

The length of the fish in meters is multiplied with a random value between 0.4 - 1 which gives the fish speed in meters per second (m/s). Having the random value between 0.4 - 1 was suggested by the external supervisor and verified in testing. These values were chosen based on results from running the simulation application with different values. Moving on, the speed in m/s is then converted to km per hour (km/h) by multiplying it with 3.6. A random value is generated for every eight locations that are generated for the fish. However, according to the external supervisor if the depth in the DST data for the current and next location has not changed much then there is likely little horizontal movement. Therefore, the random value is between 0.01 - 0.4 instead if the depth variation between the current location and the next location is 30 m or less. The resulting value is then multiplied by 24 to convert it from km/h into km per day and is then multiplied by the time step which is in days. This gives the total distance that can be covered in one iteration in km. That value is then divided by either 4 or 0.8 since the simulation uses two different OGCMs that have squares of different sizes. One is 4x4 km and the other is 800x800 m which is why it must be divided by one of the two values. Last step is to convert the final value into an integer since x- and y-coordinates are used to choose the next location, and this is done by removing all the decimals from the value.

### 3.2.11 Static Data

The simulation application spends a significant amount of time loading OGCMs from the system storage and on to the memory. This is because the DSTs used in

this project cover 7 months or over 2 years and need temperature data for every day in those periods. Therefore, data other than temperature has been removed from the OGCMs to minimise how long this takes. The data that has been removed is either not relevant for the simulation or static. For example, the OGCMs contain a value that represents the salinity of the water and takes up the same amount of space as temperature. This information is unnecessary since the DSTs does not contain salinity which is why it was removed. However, there are values like lat-$\rho$ and lon-$\rho$, seabed depth and an array that contains the depth in meters of all the vertical z-values for every (x, y) grid point. In addition, there is also an array called mask_$rho$ which is either 0 which means the (x, y) grid point is on land and is 1 when it is not. All that data is required, but static which is why it is loaded onto the memory before any of the simulations are started and remains there until the simulation application is completed.

### 3.2.12 Depth

Due to vertical movement of the DSTs used in this project the depth of each location must be checked to verify whether it is a valid location or not. This is done by comparing depth of the seabed in an OGCM grid point with depth from the current DST location. If the depth from the DST is deeper than the depth from the OGCM, the simulation moves on to check a different location. Otherwise the simulation will then iterate through all z-coordinates in the given (x, y) grid location and then choose the z-coordinate with depth value closest to actual value in the DST. This is done using a list which contains the depth of each (x,y,z) grid location (see section 3.2.11). Once the z-coordinate is chosen it stores the z-value which is an integer. The index is then used to retrieve temperature from the OGCM.

### 3.2.13 Time step

The DST data is measured every ten minutes, and OGCMs have averaged temperatures over areas within grid points. Depending on the model, each grid point is either 800x800 m or 4x4 km. It is unlikely that the simulation will be able to look at DST data for every 10 minutes because it will not be able to traverse the distance of a grid point in 10 minutes. Even if it can generate trajectories using temperature for every 10 minutes it would be time consuming. The simulation can skip 144 entries in the DST data. By doing this the simulation only looks at data for every 24 hours. This cuts out unnecessary calculations and allows the fish to travel far enough to reach a different grid point of the model.

The simulation receives a time step which is used to decide how many entries in the DST data should be skipped. This value can be given any number above zero.

Furthermore, the time step is multiplied by 144 which gives the number of entries that should be skipped and is called the tag step. There are 144 data readings every day for a DST and therefore that is the value used. Furthermore, the time step is used to increment each iteration of the simulation. The lower the time step the more complex resulting trajectories are. Unfortunately, it also reduces the number of trajectories that are generated. This is likely caused by the fact that lower time steps lead to the simulation having to check more locations before it reaches the recapture location. Therefore, it has more opportunities to fail. Also, trajectories will not be able to travel very far each iteration. This may allow the simulations to travel into the fjords, but they may not be able to exit since a limited number of directions are checked and most of them could be blocked by land.

### 3.2.14 Implementation of Ocean Current

At the suggestion of the external supervisor the ocean current will be used in the simulation to determine where the fish is more likely to travel. This information is extracted the same way as temperature and uses the same coordinates. The ocean current is represented by two values called u and v. The value v represents the north and south ocean current, a positive value means north and negative means south. For u it is east and west, positive for west and negative for east. The value itself also indicates the strength of the current. The question is, how should the current influence the simulation? Should simulation follow or go against the current? It is uncertain if either is the correct approach and therefore both will be evaluated. Following the current means the simulation is more likely to choose locations that the current is moving towards. Going against the current means it more likely to choose location the current is coming from.

### 3.2.15 Choosing Optimal Ocean General Circulation Model

There are two OGCMs available, the Nordic Seas model and the Norwegian Sea model. The Norwegian Sea model has a higher resolution which makes it the preferred option, but this model can only be used for locations in the waters around Norway. Having the simulation switch between models when necessary could improve the number of trajectories. Testing with DST 1664 made it clear that using only the Nordic Seas model was not good enough because it was not able to produce any complete trajectories. However, when the model of the Norwegian Sea was used instead, it was able to complete but could not travel to the Barents Sea because the model only cover the waters around Norway.

Switching between the Nordic Seas and Norwegian Sea models is not that simple. Currently the simulation uses the x- and y-coordinates of the Norwegian Sea

model to decide where it is. The x- and y-coordinates in the Nordic Seas model are not equivalent since both have different resolutions. Therefore, the latitude and longitude will have to be used instead. This means that simulations that switch from the Norwegian Sea will have to go through and compare its current latitude and longitude with all the latitude and longitude values in the Nordic Seas model. The double list of latitude and longitude are sorted so a search algorithm, e.g. binary search, could be used. In the worst case it will have to search through 2 347 004 x- and y-coordinates to find the correct coordinates.

The data is static, so it would be less computing intensive to simply index them to each other. Create two lists that take the x- and y-coordinates of one model and returns the x- and y-coordinates for the other model. So, one list for the model of the Norwegian coast and the other for the Nordic Seas. This is what was done in this thesis. The two lists were stored as NetCDF files that the application loads into the memory at the start of the simulation. Doing it this way means the simulation only has to check one value when converting instead of over two million.

This implementation does have an issue. Every x- and y-coordinate in the Nordic Sea model is indexed to a x- and y-coordinate Norwegian sea model. Even the coordinates that are outside the area covered by the Norwegian Sea model. Therefore, a (x,y) grid point could be wrong when converting to the Norwegian Sea model. The solution to this is to take latitude and longitude in the (x, y) grid point that is to be converted and compare to the latitude and longitude of the converted point. Then, if the difference between the latitudes and longitude is less than 0.1 the conversion was successful. Otherwise the trajectory is terminated.

Switching between the two models is not always necessary. The DSTs 742 would not benefit from this because, according to the external supervisor, it moves along the Norwegian coast. The model with the highest resolution already covers this area, therefore the simulations of this DST will only use the Norwegian Sea model.

DST 1664 was released 02.04.2004 and recaptured 19.05.2006. It is unlikely that it spent those two years in Lofoten where the release and recapture locations are. Therefore, weighting the fish towards the recapture location will not work because that will prevent it from leaving Lofoten. Being in the ocean for so long means there is no way of knowing if it travelled to the Barents Sea or not. Therefore, the simulations of this DST will use both the Nordic Seas and Norwegian Sea models.

### 3.2.16 Reading NetCDF

Originally the SDSLite NuGet package would be used to read NetCDF files in the program. Unfortunately, it was outdated and therefore unable to access the correct dll files containing all the NetCDF functions that were needed. To solve this issue the SDSLite project was cloned from its Github repository [43]. SDSLite use a NuGet package called Dynamic interop to import all functions from the dll. The package was created to import functions from a "unmanaged" dll. However, this did not work and resulted in a PInvoke exception because the functions from the dll were not called correctly. The solution was to not use Dynamic interop, but rather import all the functions using the code below:

```
[DllImport(@"\path{C:\Program Files\netCDF 4.6.1\bin\netcdf.-
dll}", CallingConvention = CallingConvention.Cdecl)]
public static extern int nc\_set\_chunk\_cache(IntPtr size,
IntPtr nelems, float preemption);

[DllImport("netcdf.dll", CallingConvention = CallingConventi-
on.Cdecl)]
public static extern int nc\_open(string path, CreateMode mo-
de, out int ncidp);
```

The first line of code has the full address to the NetCDF.dll, but the other DllImport lines simply write the dll name instead. This is because the other lines know where it is because of the first. The two lines of code with DllImport are there to make sure that the function, which is right below it, gets imported correctly.

### 3.2.17 DSTs used

Based on the recommendation from the external supervisor this paper will focus on two DSTs. First is DST 742 which was chosen since the release and recapture locations are far apart and it has been out for 195 days. It was released south east of Svalbard and recaptured in Lofoten. It was released 26.08.2003 and recaptured on the 13.03.2004. Based on this data it is likely that the fish travelled straight from the release to recapture location which makes evaluating the trajectories easier. It collected DST data for about seven months and will not be as time consuming as other DSTs that have collected data for a longer time. The second is DST 1664 which collected data for far longer time than DST 742. It was released on the 02.04.2004 and recaptured the 22.12.2006, meaning it collected data in the ocean for roughly thirty-two months. Furthermore, the release and recapture locations are both in Lofoten. This means that the weighting that was used for DST 742 will not work for DST 1664. Therefore, simulating the trajectory for this DST will be far more complicated and time consuming. There is also the possibility that this could reveal

shortcomings of the simulation application that were not apparent in the trajectories generated from DST 742. Moreover, since DST 1664 has data for over two years it is difficult to evaluate the trajectories that will be generated from it. Trajectories generated from this DST will prove whether the simulation is of any value. Also, the DST only contains temperature and depth data from the release date to 19.05.2006. Without those seven months of data the merge algorithm will not work since it runs simulations from the release and recapture location. However, the general algorithm should still work, since it only runs simulations from the release location.

## 3.3 Visualisation Application

The visualisation application of the framework is created in Unity3D with a scene for user interaction and background implementation for managing requests from the user.

### Unity3D Scene

The Unity3D scene contain the environment and menus of the application. The environment, "Map" is an imported satellite map provided by Mapbox API, and the three displays; "FishTrajectoryDisplay", "ChangeParameterDisplay" and "DataOnMarkerDisplay" are the interactable menus for the user.

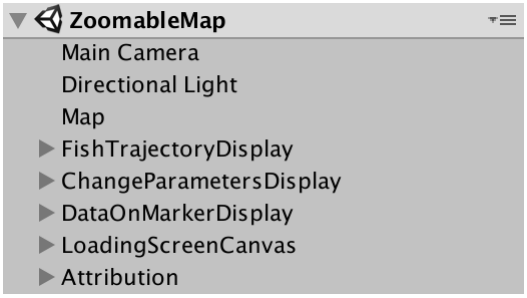



Figure 3.8: The parent/child structure of the application

The application is created with the parent/child structure, where the parent causes all children to move the way the parent does, but moving the children does not have any effect on the parent. Each location in a fish trajectory is created as a child of the scene. This means that the locations can be modified individually without affecting the scene or other locations.

### Mapbox

An imported satellite map from Mapbox API is the backbone of the visualisation application. Along with providing maps, the Mapbox API consists of functionality that allows customisation of maps, as well as adding elements to maps. Mapbox also has a build in zoom and it updates the location of the coordinates as the user navigates around the map.

To make the map suitable for visualising fish trajectories in the Nordic Seas, the script "SpawnOnMap.cs" is modified. Since the fish trajectories are centred around Norway and the Barents Sea, the initial displayed location of the map is altered to the waters around Norway.

Figure 3.9: Satellite map from Mapbox API in Unity.

The locations in a fish trajectory can be displayed through adding custom markers that appear on the map through transforming GPS coordinates into x- and y-coordinates for the Unity3D scene. Mapbox API provide this functionality, along with functionality to scale and add as many markers necessary.

The "SpawnOnMap.cs" script initially drew locations on the map at run time, asking the user to manually decide their location. The simulation algorithm store fish trajectories by writing latitude, longitude, temperature and depth in text files, where each line represents a new location in the trajectory. The script is modified to read the text file and add each line to a list. Instead of listening for user input, the script will now draw each object in the list as locations in the fish trajectory. The user can change which trajectory drawn by choosing a file from one of the two drop-down menus in the Fish Trajectory Menu. The application will then remove the previous trajectory from the view, and both read and draw the new file on map.

### Fish Trajectory Menu

The Fish Trajectory Menu read files and presents content to the user, as well as allowing the user to modify how content is presented.


Figure 3.10: The Fish Trajectory Menu

Figure 3.10 shows the menu presented when the application is started. First, the application read a file named "AvailableTags.txt" that contain all the available DSTs the simulation application can execute. This file must exist to use the application because a DST is only available if both the DST data and the OGCMs are available. Each of the available DSTs are placed in a drop-down menu named "Choose fish tag" that change the directory path to the path of the chosen DST.

When a DST is chosen, the two drop-down menus "Paths close to capture point" and "Paths not close to capture point" are updated. They read two separate directory folders containing files of potential fish trajectories, where one folder contains trajectories that end close to the recapture location of the fish and the other folder contains trajectories that do not end close to the recapture location. The drop-down menus contain all the fish trajectories and when a fish trajectory is chosen, the view is updated to display the new trajectory.

Trajectory files from an execution of the simulation application will override previous results on a given DST. This ensures that all the available trajectories derive from the same parameters, which makes the comparison of trajectories more accurate in terms of finding common migration patterns.

In order to let the user modify how trajectories are presented, toggle buttons are created. This is done in order to visualise trajectories without crowding the view with all the information at once. Toggles work as switches where behaviour are programmed to their two conditions, checked and unchecked. The program listens to changes on each toggle and respond based on their condition. Each of the four toggles are described in table 3.1 - 3.4 below.

| **Draw whole trajectory** | **Description** | **Purpose** |
|---|---|---|
| **Checked** | Draws every location in trajectory | To see the full fish trajectory |
| **Unchecked** | Draws release and recapture locations | To see where the fish was released and recaptured |

Table 3.1: Toggle for drawing the whole trajectory

| **One by one location** | **Description** | **Purpose** |
|---|---|---|
| **Checked** | Draws one location every 0.05 second | To see the movement and direction between each location |
| **Unchecked** | Draws every location in fish trajectory at once | To more easily compare similarity of fish trajectories |

Table 3.2: Toggle for drawing one by one location

| **Index on locations** | **Description** | **Purpose** |
|---|---|---|
| **Checked** | Shows each location with its index | To see the direction the trajectory is moving |
| **Unchecked** | Shows each location without its index | To make it easier to distinguish between individual locations |

Table 3.3: Toggle for showing index of each location

| Draw lines | Description | Purpose |
|---|---|---|
| **Checked** | Draws lines between each location | To see connection between locations |
| **Unchecked** | Remove lines between each location | To make it easier to distinguish between individual locations |

Table 3.4: Toggle for drawing lines between locations

### Change Parameters Menu

The Change Parameters Menu is created to let the user change parameters on the simulation algorithm. The menu is reached through clicking the "Change parameters" button from the Fish Trajectory Menu. Since the menu covers most of the available space on the screen, the button changes the state of the menu to either display or hide it.


Figure 3.11: The Change Parameters Menu

When the application is started, it read a file named "Setup.txt" that contains the parameter values from the previous execution of the simulation algorithm. These parameters are used as placeholder values in the Change Parameters Menu to help the user recall previously used parameter values. If the file does not exist, the placeholder values are set to zero until the user runs the simulation algorithm that store the parameter values in the "Setup.txt" file. Figure 3.11 display the menu with values from an earlier execution of the simulation algorithm.

The parameters are explained in table 3.5 below. Each of them affects the result from the simulation application, either by the number of trajectory results or by increasing or decreasing the execution time of the simulation application.

| **Parameter** | **Description** |
|---|---|
| Fish length (meters) | Factor for determining swimming speed |
| Days between each tag data | How often the simulation algorithm should read DST data and calculate locations |
| Number of simulations | Amount of individual fish trajectories |
| Allowed margin of error on temperature | Difference between temperature in DST data and in OGCM for deciding a potential location in the fish trajectory |
| Allowed margin of error on depth | Difference between depth in DST data and in OGCM for deciding a potential location in the fish trajectory |
| Weighting of trajectory towards recapture point | Percentage of how much each location should be weighted towards the direction of where the fish was recaptured |
| Number of new locations to check in each location | How many potential locations that are calculated for each location |
| Choose algorithm | Lets the user choose which simulation algorithm to run |

Table 3.5: Explanation of the parameters for running the simulation application

When the button “Run” is clicked, a terminal window is launched containing information about the progression of the simulation application. When the simulation application is completed, the terminal window will provide information about the results from the execute and then close itself. The “Setup.txt” has now been changed to the new parameters, the new trajectories have replaced the old trajectories, and the map is automatically updated to display one of the new trajectories.

### Data Menu

The trajectories resulting from the simulation application store latitude and longitude, as well as temperature and depth from the DST data and the OGCM in each location. The Data Menu displays the temperature and depth information for every location in a fish trajectory. Figure 3.12 display the information for marker 77, as well as the difference between the values. The Data Menu also display the distance and average speed to a previous or next location in a trajectory. To easily highlight the next or previous location, the buttons "Previous Marker" and "Next Marker" change the current location and updates the temperature, depth and distance viewed.

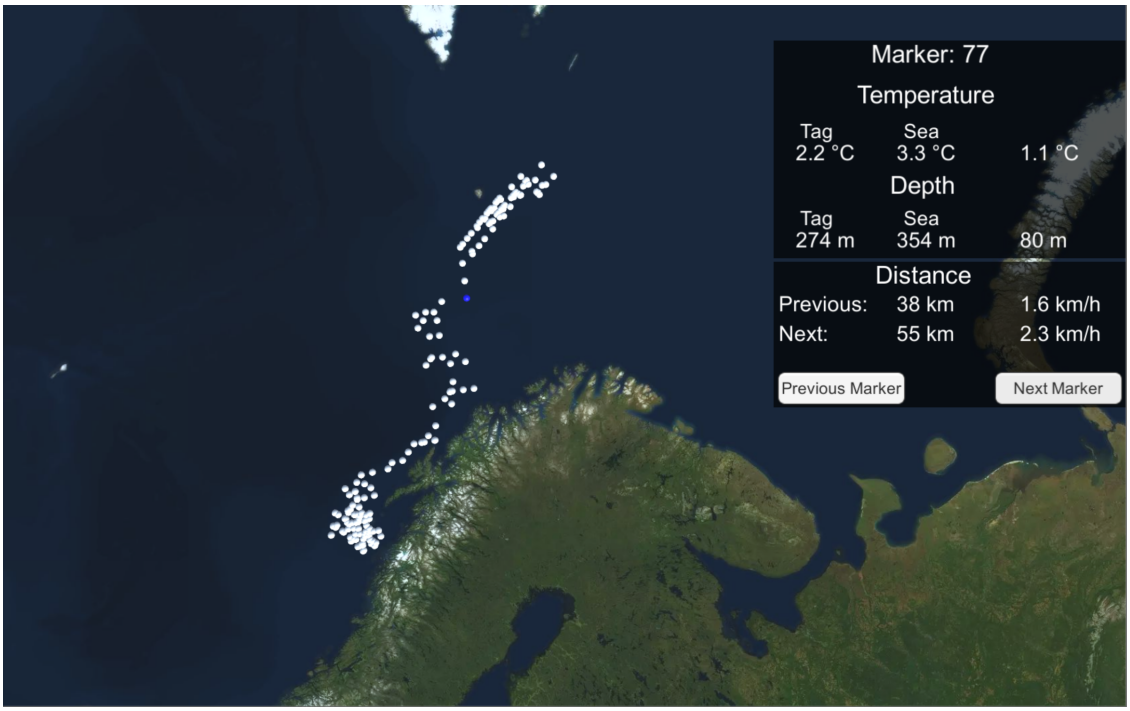



Figure 3.12: The Data Menu

# Chapter 4

# Results

The results from the simulation application will be presented in this chapter. The results derive from a computer with a Ryzen 5 1600 CPU, 16 GB of RAM, 1 TB SSD and a 2 TB HDD. Results from simulations of DST 742 used the Norwegian Sea model and results from DST 1664 used both the Norwegian Sea and Nordic Seas models. The results shown will be the time it takes for the simulation application to finish, and the number of trajectories generated. A high number of trajectories is considered good since it means that the parameters, e.g. margin of error on temperature can be lowered. Table 4.1 shows the parameters and the values used in most of the test that were run. If a test uses different values for any of the parameters, it is specified. Furthermore, fish length is unique to each DST and is therefore different. Also, the weighting is different for the DSTs.

| **Parameter** | **Values** |
|---|---|
| Fish length (meters) | 0.82 meters for DST 742<br>0.65 meters for DST 1664 |
| Days between each tag data | 1 day |
| Number of simulations | 10 000 |
| Allowed margin of error on temperature | 1.2 °C |
| Allowed margin of error on depth | 30 m |
| Weighting of trajectory towards recapture point (0 - 1) | 0.75 on DST 742<br>no weighting on DST 1664 |
| Number of new locations to check in each location | 30 |

Table 4.1: Parameter values used when testing the simulation application

## 4.1 Research Method

In order to answer the research questions, data on performance and usability had to be collected. The performance of the framework was tested by executing the simulation application multiple times with the same parameters to see if the execution time or the number of trajectory results varied. To collect data on feedback, the spiral development model explained in section 1.7 was used as the development methodology. How the methodology was used is described below.

The project followed a 15-step spiral development model (SDM). In order to iterative over each phase three times, the number of steps was determined to be 15. This resulted in more than one prototype, which made it easier to collect data and match expectations of experts. Figure 4.1 display the SDM along with an explanation of what was done in each step.

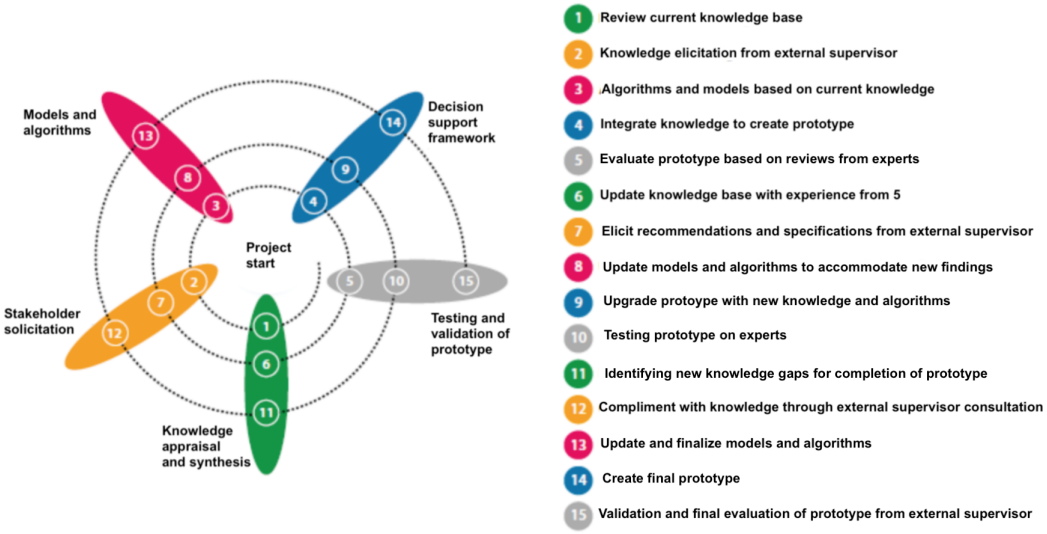



Figure 4.1: The spiral development model followed for creating the framework

### Phase I: Knowledge Appraisal and Synthesis

In this phase, the goal was to review current knowledge base. In step 1), a problem description was used to find unknown terms and fields of studies required to solve the problem, e.g. DST, OGCMs, migration patterns. Step 6) and 11), identified knowledge gaps based on feedback and tests from a selection of experts. Knowledge acquired in this phase were discussed at the end of each step to decide if the knowledge could be used to further improve the solution. Instead of only acquiring knowledge on topics that would expand the functionality, knowledge for improving resulting fish trajectories was often prioritised. Ocean current is an example of knowledge investigated in order to potentially improve resulting fish trajectories

(see chapter 3.2.14).

#### Phase II: Stakeholder Solicitation

In Phase II the goal was to elicit knowledge from the external supervisor that could improve understanding of topics and fields of studies. This could be links to articles in order to gain a deeper understanding of knowledge acquired in Phase I, access to internal servers to download data, e.g. OGCMs, or connecting us with experts with deeper knowledge on topics, e.g. oceanographer to understand data within OGCMs.

#### Phase III: Models and Algorithms

Based on knowledge acquired from Phase I and II, the design and algorithms for the solution is created in this phase. In step 3), the design for the simulation application and the visualisation application was created, along with algorithms for reading NetCDF files and generating trajectories. In step 8) and 13), the previous algorithms were updated based on new knowledge acquired and algorithms for new functionality were implemented.

#### Phase IV: Decision Support Framework

This phase integrates knowledge and algorithms into a prototype that can be tested by experts. This phase also eliminates potential functionality and improvements found in phase I and II, that do not improve the current solution. Turbulent ocean current affecting where fish traverse was knowledge learned late in the project (see chapter 3.2.14). A solution with an algorithm that found locations within ocean currents did not affect resulting trajectories, but it did result in longer execution time of the simulation algorithms (see chapter 4.4.1). This resulted in assembling a prototype without the algorithm for finding turbulent ocean current.

#### Phase V: Testing and Validation of Prototype

In Phase V, the prototype created in Phase IV is tested and validated on a selection of experts chosen by the external supervisor. They are all employees at the IMR, and they are experts in different fields of studies, e.g. biology, mathematics, oceanography. In step 5), the first prototype was presented to the experts, including the external supervisor, at a meeting where they received a presentation and a demonstration of how the framework was created. They gave comments on improvements and functionality that they found necessary in a solution for generating and visualising fish trajectories. In step 10), a survey was performed on the experts where they used the prototype without guidance, and then answered questions on its usability and functionality. The approach and the results from this survey are explained in chapter 4.2 below. In step 15, the final prototype was updated with functionality

and variable names from the expert survey and presented to the external supervisor, where the future of the framework was discussed.

## 4.2 Expert Survey

The expert survey was performed in step 10 of the spiral development model. This made it possible to add functionality and improve usability before submitting a final prototype. This section explains how the expert survey was performed and the collected results.

### 4.2.1 Approach

#### Selecting the Experts

The external supervisor has chosen a group of experts suited for reviewing and providing feedback for this project. The experts are employed at the IMR, and have experience related to geo-location of fish. They have investigated geo-location of fish in different fields of studies, e.g. mathematics, biology, oceanography and ecosystems. The experts were available for questions on topics related to their field of study and provided feedback on two prototypes.

#### Setup for Testing the Framework

Three domain experts tested the prototype from step 10 in the SDM. The domain experts consisted of a biologists who is responsible for the DST used, an oceanographer who created the merge algorithm [6], and a researcher of ecosystems and migration patterns.

| Parameter | Recommended values |
|---|---|
| Fish length (meters) | 0.6 - 0.8 m |
| Days between each tag data | 1 - 4 days |
| Number of simulations | 5 000 - 15 000 |
| Allowed margin of error on temperature | 1.2 - 1.5 °C |
| Allowed margin of error on depth | 10 - 30 m |
| Weighting of trajectory towards recapture point (0 - 1) | 0.65 - 0.90 |
| Number of new positions to check in each position | 8 - 14 |
| Choose algorithm | General / Merge |

Table 4.2: Recommended parameter values for running simulation algorithm

To perform a test without guidance from developers, a summary of what the framework aims to accomplish was created along with a brief introduction to different functionality in the visualisation application. A step-by-step guide on how to run the simulation application from the visualisation application was also created, together with table 4.2 containing recommended value ranges for the parameters.

A survey was created in order to gather information regarding available functionality, variable names, usability, and improvements. The results from the survey is presented in chapter 4.2.2, and discussed in both chapter 5.1.3 and 5.1.4.

### 4.2.2 Results

The experts used the framework for 30 minutes and answered the survey afterwards. In order to make the experts understand the ongoing process behind the visualisation application, it was explained to them how the simulation algorithms find trajectory results. This was done while the experts were waiting for trajectory results from an execution that they had launched with the help from the step-by-step guide and the parameter values from chapter 4.2.1 and table 4.2. Since the user interacted with the visualisation application of the framework, the questions and feedback in the survey was mostly concerning the front end of the framework and not back end improvements on how fish trajectories are generated.

The results from the survey can be found in Appendix A, and they provide feedback

on the usability of the application. Question 1), 3) and 5) go through all the functionality and parameters in order to determine how well functionality are explained by their names. The results from these questions were positive, and the experts suggested improvements to some of the names, see question 2), 4) and 6). Question 7) discover how well information is presented in the execute window that is launched when the user starts a simulation from the visualisation application. This execute window display information on progression of the simulation application, and it did receive suggestions on how to improve it. Question 10, "how well does the application present fish trajectory results?", received top score from the experts. The next question, "considering the parameters used and the approach for simulating fish trajectories, how likely is it that the trajectories are realistic?", received mixed answers since the experts cannot with certainty decide if the results are realistic. The last question, "would you use a framework, like the one tested, to simulate and visualise DST data?", showed that two out of three would use a framework like this.

## 4.3 Visualisation Application

The results from the expert survey gave ideas on how to improve the visualisation application. A discussion on the suggestions that were made and how they were prioritised, is presented in chapter 5.1.3. New functionality was included, as well as more precise description of variables.


Figure 4.2: Final prototype of the visualisation application

Figure 4.2 display the final prototype presented to the external supervisor in step 15 of the spiral development model. Based on the feedback from the experts, the Data Menu now display total amount of locations as well as the date for each location in the trajectory. The headline "Distance" was adjusted to "Distance between positions" to make it more understandable.

In the Fish Trajectory Menu, new functionality was added. A toggle button named "Several paths", allow the user to display several trajectories at the same time. This makes it easier to see how much trajectories vary from each other. A button named "Save path", and a drop-down menu with saved paths was also included. The possibility of saving trajectories allow the user to analyse trajectories with different parameters. The user can click the "Save path"-button and access all saved trajectories in the drop-down menu "Saved paths".

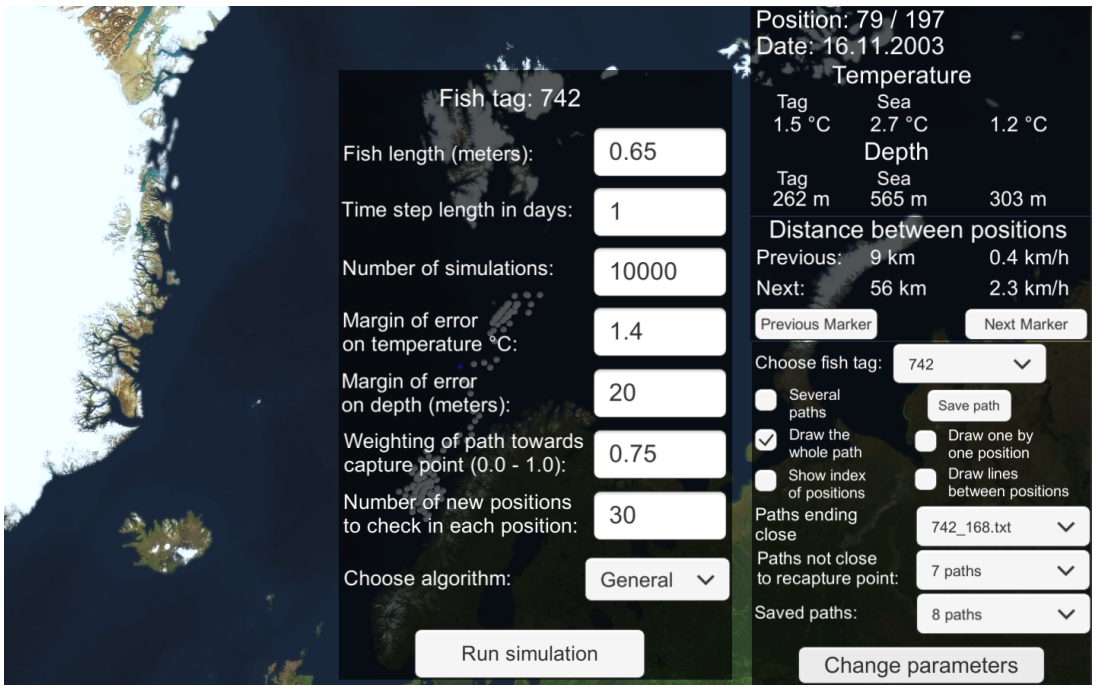



Figure 4.3: Final prototype of the visualisation application, including Change Parameters Menu

In the Change Parameters Menu, Figure 4.3, several fields were renamed to make it easier to understand the purpose of the parameters. The previous prototype ran out of space in the field "Number of simulations" when the number exceeded 10 000. All the fields were therefore increased in size.

## 4.4 Simulation Application

### 4.4.1 Ocean Current

After checking the results of implementing ocean current, the resulting trajectories are not noticeably different. It made no difference whether the simulations followed the ocean currents or went against them. The number of trajectories was unaffected and same for the locations in them. However, the run time was significantly increased since adding ocean current data to the NetCDF files with OGCM temperature data, tripled the size of each file. This led to increased run times and a storage constraint. Figure 4.4 show the increased run time after adding ocean current. Using an HDD more than doubled the run time while with an SSD increased with a little more than two minutes.

Running the simulation without implementing the ocean current data on an SSD the CPU utilisation was well over 80% for the first half of the simulation application. On the second half the utilisation was between 30-50%. This is a result of simulations being terminated as the simulation application runs, which leads to less work for the CPU. However, with the ocean current implemented the CPU utilisation was consistently lower and would often drop and then spike, especially when using an HDD. This was likely because it had to wait for the NetCDF files to be loaded from the system storage and onto the RAM.

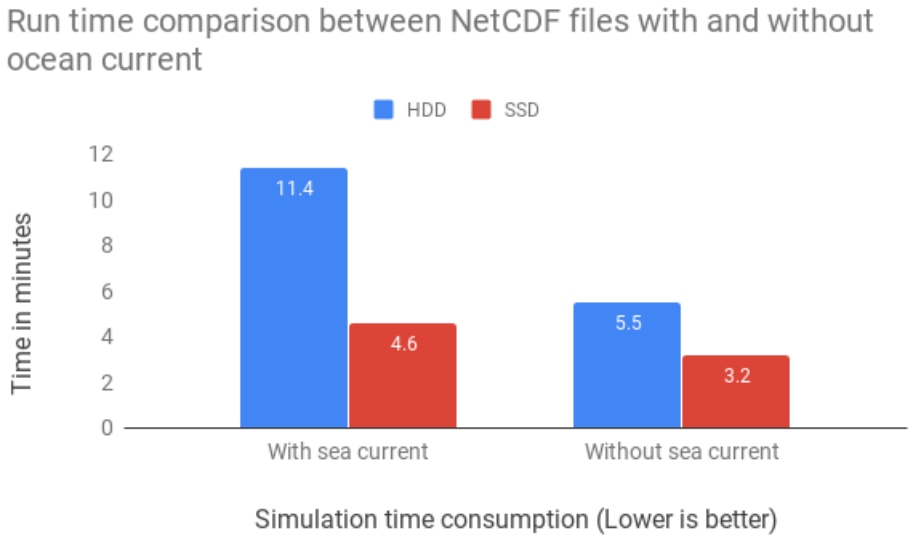



Figure 4.4: Comparison of the run time of both the algorithms with and without the ocean current implemented.

In conclusion, the fact that the ocean current implementation had no effect on the trajectories and increased the run time, it was removed from the solution.

### 4.4.2 DST 742

There are certain patterns that appear when looking at trajectories generated by simulating of DST 742. In the beginning, all trajectories are within the same area for roughly 70 days before any of them start to travel towards Lofoten regardless of weighting. According to "Migratory behaviour of north-east Arctic cod, studied by use of data storage tags" [44], tagged fish behave differently the first fourteen days after release, and return to "normal" behaviour afterwards. The article states that the reason for this is that the fish have ruptured their swim bladder, which regulates its pressure. This could mean that the fish stays within one area because it needs to recover before it can traverse towards Lofoten.

#### General Algorithm

When generating trajectories using the general algorithm there is not much variation when looking at the locations in the trajectories. This is likely because all the trajectories must get far enough south before the temperature in the DST reaches 8.0 °C. Afterwards, all the successful trajectories spend the remaining steps within Lofoten because of the high level of weighting that must be applied for the trajectories to able to complete. Therefore, it is likely that the temperature is so high because the fish has entered or is close to Lofoten.

#### Merge Algorithm

Similar to the general algorithm, the merge algorithm also does not show much in terms of variation between each trajectory at first glance. However, it does generate some trajectories that are different from the general algorithm. This is likely because trajectories are created by combining two trajectories where one started in the release location and the other in the recapture location. Then both were weighted towards the location it did not start in. Nevertheless, most of the trajectories are very similar to the once produced by the general algorithm, but there are trajectories that travel further west and do not follow the Norwegian coast as closely. It could be caused by the weighting of the backwards trajectories towards the release location.

### 4.4.3 DST 1664

The trajectories generated from the DST 1664 have significantly more variations than DST 742. This is because both the release and recapture locations are so close together and it has been out for over two years. This means that the current weighting towards the recapture location will not work. Therefore, the simulation

was run with no weighting at all, however there were no apparent and consistent patterns that emerged from the trajectories.

### General Algorithm

The trajectories generated through simulations of the general algorithm are concentrated around Lofoten and do not travel far into the Barents Sea. Comparing the locations in the trajectories with the area covered by the Norwegian Sea model indicates that few trajectories ever switch over to the Nordic Seas model. Furthermore, this also means that it is likely that most of the trajectories that do switch to the Nordic Seas model are terminated. Alternatively, it may be that the trajectories rarely change from using the model of the Norwegian Sea to the one of the Nordic Seas. For a trajectory to switch there must be no possible location found using the current model. It could be that none of the trajectories that are able to complete ever switch to the Nordic Seas model because it always finds at least one valid location. However, the trajectories that do switch might actually travel into the Barents Sea, but are unable to find their way back to Lofoten before the temperature rises too high. The solution to this would be dynamic weighting that changes based on the time of year. It was determined that none of the completed trajectories travelled to the Barents Sea because it will not switch unless it is unable to find any valid locations. Therefore, a check was implemented that would switch the models that was being used if the trajectory was far enough north. This was done by checking if the latitude was greater than 71° and if the longitude was greater than 25°. Then it would switch to the Nordic Seas model and the opposite is true for the Norwegian Sea model. This did lead to some trajectories visiting the Barents Sea, but none stayed for more than six days and the locations were barely within the Barents Sea.

### Merge Algorithm

Unfortunately, this algorithm did not initially work since the last recorded data with depth and temperature in this DST is from 19.05.2006 while the recorded recapture location is from 22.12.2006. This leaves a gap of seven months with no data. Therefore, to continue the work on this DST it was decided that the last registered date in the actual DST should be used instead of the recapture date.

With the time step to 2 days and the error margin on temperature at 1.2 °C the merge algorithm is unable to find any new valid locations. Furthermore, increasing or decreasing the time step had no effect. This is only an issue for the backwards trajectories while the forwards trajectories can complete without any issue. A part of the reason for this is that none of the locations it checks are valid since there is land in between the new and the current location.


Figure 4.5: Map of where DST 1664 was released (1), recaptured (2) and where the recapture locations was moved to (3).

This is caused by the recapture location of this DST being on land, deep within Lofoten as illustrated in figure 4.5 where the recapture location is marked with number 2 and the release location is marked with the number 1. Because of this there is plenty of land that can block the directions that the simulation checks. Therefore, the simulation of the backwards trajectories cannot get any further than the first iteration. Taking a closer look revealed that the simulation found two locations that were not blocked by land and met all other requirements except the temperature margin of error of 1.2 °C. As a result, the locations where discarded because the margin of error on temperature was too low. Once the margin of error on the temperature was raised to 2.2 °C it was able to go beyond the first iteration. With this change the simulation was able to generate roughly 4,500 backward trajectories that were ready to be combined. Unfortunately, using such a high margin of error on temperature meant that generating the backward trajectories had an execution time of 15.52 minutes. Without a high margin of error on temperature for the backward trajectories the simulation application is unable to run this DST using the merge algorithm. However, the forward trajectories can be generated using a lower margin of error on temperature.

The recapture location is less accurate than the release location according to the external supervisor and that is why the general algorithm has a radius around the recapture location. If the trajectory ends close enough to the recapture location it is considered acceptable. Based on this, moving the recapture location of DST 1664

from location 2 in figure 4.5 to location 3 should still give valid trajectories and avoid issues such as land blocking most of the directions. Doing so lead to the algorithm being able to run the backwards trajectories of the algorithm without any issues and using the same margin of error on temperature as the forward trajectories.

### Limitations of the Ocean General Circulation Model

After reviewing the visual representation of the trajectories generated from DST 1664 it appears that all the trajectories are being prevented from travelling further west. This is seen by comparing the area covered by the Norwegian Sea model and the generated trajectories. The red line in figure 4.6 shows roughly were the area covered by the Norwegian Sea model ends and this makes it clear that there are several locations in all the trajectories that travel close to the western border (red line) of the model, but never cross it.

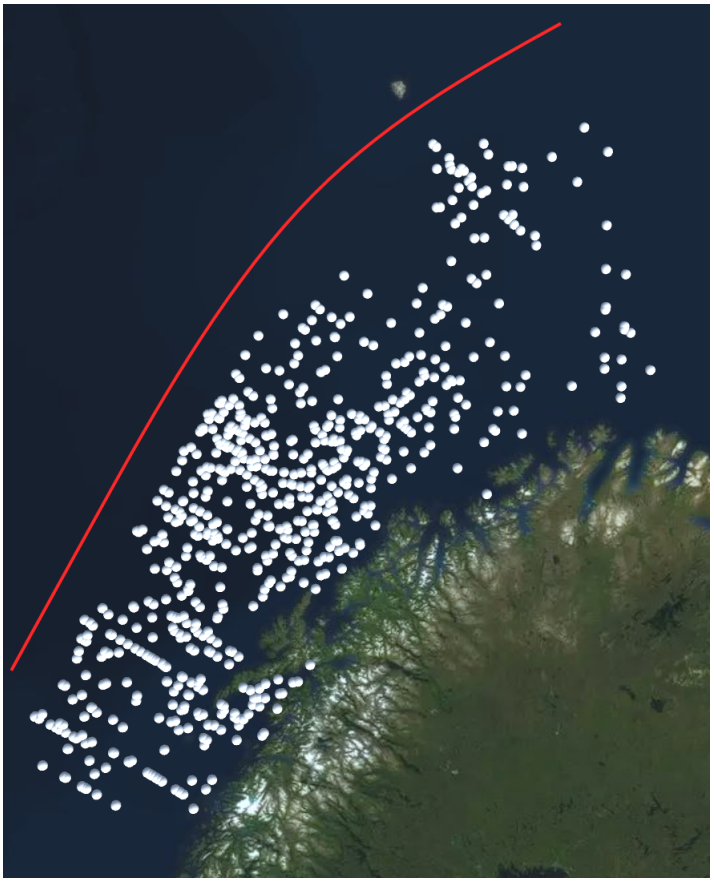


Figure 4.6: An example trajectory generated from the DST 1664 using the merge algorithm. The red line represents the western border of the area covered by the Norwegian Sea model.

This is likely caused by the fact that the simulation will not switch to the Nordic Seas model unless it is unable to find any new valid location using the Norwegian Sea model or far enough north. To counter this, the simulation could implement a

similar condition as the one implemented for trajectories that are far enough north to decide which model to use.

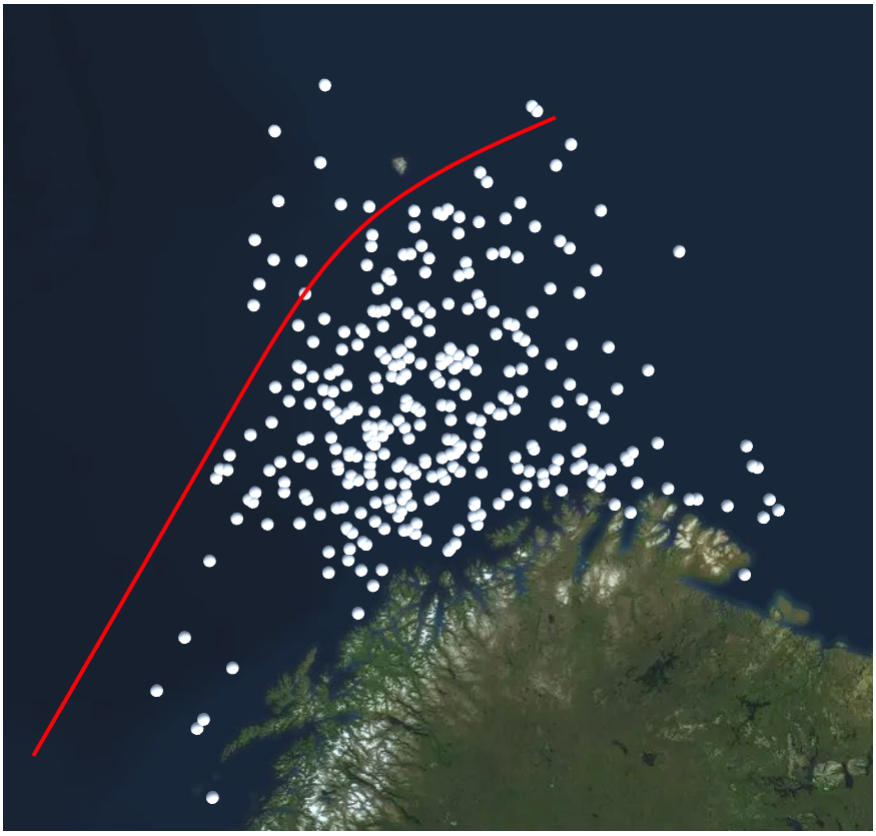


Figure 4.7: An example trajectory generated from the DST 1664 using the general algorithm. The red line represents the western border of the area covered by the Norwegian Sea model.

Figure 4.7 shows one of the trajectories that did switch which model it was using when close to the red line so that the trajectory could travel further west. However, there were extremely few trajectories that ever did cross the red line and the ones that did were categorised as unacceptable since they did not end close to the recapture location. The other completed trajectories never crossed the red line and that could be because the Nordic Seas model did not have temperatures that were valid and therefore many of the trajectories that did switch were not able to complete.

### Dynamic Weighting

The DST 1664 has its release and recapture locations next to each other as seen in figure 4.5. Therefore, weighting the trajectory towards the recapture location will not work and a different approach is required. This is dynamic weighting which means that the trajectory will be weighted away from the recapture location in Lofoten when the simulation is reading the DST data from May. Then, when reading the DST data from November the trajectory will be weighted towards the recapture

location. These months were selected based on the recommendation of a biologist employed at the IMR. Unfortunately, this weighting reduced the number of trajectories produced and none of the trajectories that were generated were close to the recapture location. Furthermore, none of the trajectories displayed any unique characteristics compared to other trajectories generated without any weighting.

### 4.4.4 Switching Between Ocean General Circulation Models

Figure 4.8 and 4.9 compare the results of the simulation application with and without switching OGCMs. All the results are derived from running half of the data within DST 1664 through the general algorithm. The time step was set to four, meaning the simulation only looked at DST data for every four days. The algorithm with switching (see section 3.2.15) uses both the models for the Nordic Seas and the Norwegian Sea. The algorithm that did not switch, used the Nordic Seas model which has a significantly lower resolution compared the model of the Norwegian Sea. All settings were the same and the simulation was run three times and then an average was taken of the number of trajectories results and time from all three runs.

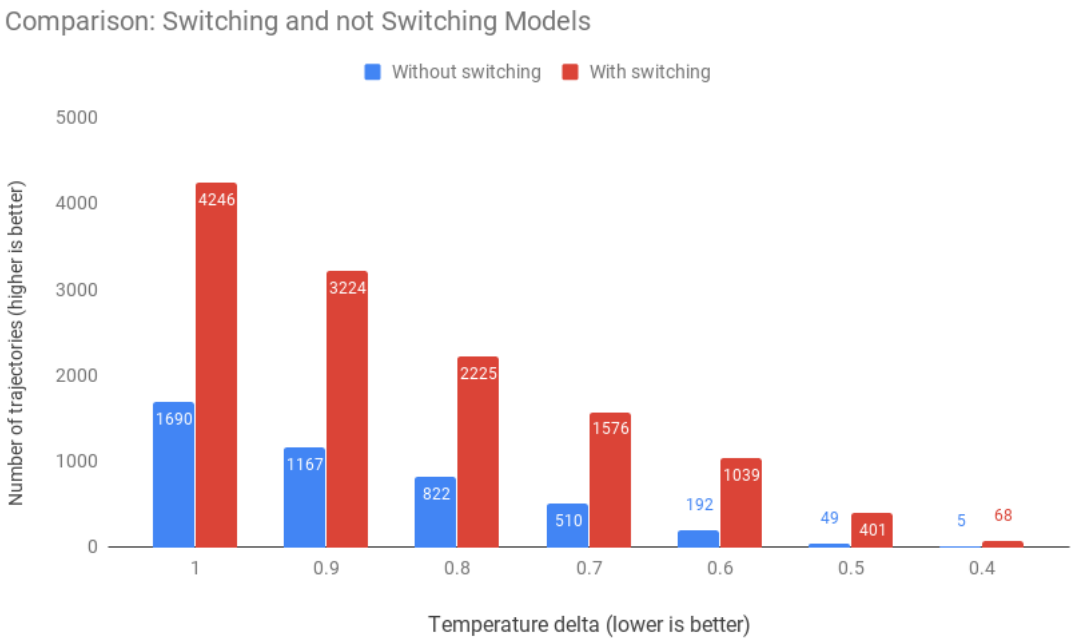



Figure 4.8: These simulations were run with only half of the data in DST 1664.

Figure 4.8 shows how many trajectories created by both methods with different margin of errors on temperature ranging from 0.4 - 1.0 °C. A low margin of error on temperature is considered good since it increases the likelihood of the selected location being the actual location of the fish. However, a low margin of error on

temperature means that simulations will find fewer valid locations since the temperature difference must be lower. Consequently, this makes it is more likely that a simulation will find zero valid locations, which means that fewer trajectories are generated. Regardless of what the margin of error on temperature is set to, the switching method shows a significant improvement over using only one model in the number of trajectories that are generated. The difference in number of trajectories between the two methods only increases as the margin of error on temperature is reduced. For example, with a margin of error on temperature set to 1 °C the switching method produces over 2.5 times the number of trajectories that the method produces without switching. Furthermore, with the margin set to 0.4 °C gives over 13 times the number trajectories.

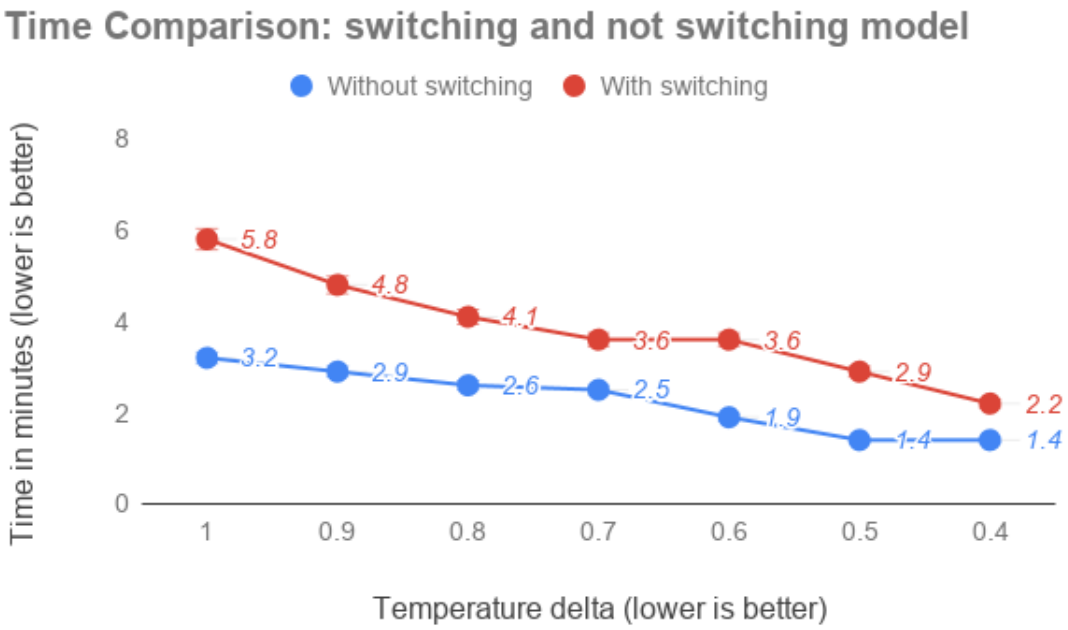



Figure 4.9: Illustration of the time reduction over time

Unfortunately, this does lead to an increase in the run time of the simulation application. This can be seen in figure 4.9. However, the number of trajectories that are generated also increases with run time. As the margin of error on temperature is reduced, so is the difference between the run times of both the methods. Furthermore, when the margin of error on temperature reaches 0.7 °C, the run time difference is not as significant.

## 4.5 Performance

### 4.5.1 Time Consumption

Early in the development of the framework, OGCMs were not loaded directly into the simulation application. Instead it called an external Python script that would return the temperature from it. Each call needed about 100 ms to complete, and it was done eight times for every simulation in every iteration of the simulation application. For example, if the simulation application is run with 100 simulations, 48 iterations and the number of new locations to check in each location is 8 this is the result:

$$\frac{48 \text{ iterations} \times 100 \text{ simulations} \times 8 \text{ locations to check} \times \ 100 \text{ ms}}{60000} = 64 \text{ minutes}$$

Assuming none of the simulations are terminated this is the minimum run time that is required since only the time that is required to extract temperature values from the OGCMs is considered. This was done because the OGCMs could not be loaded into the C# project since SDSLite cannot read files that are that large.

Later on in the development, the models were loaded into the simulation application. The problem was that they were too big. Therefore, they were divided into smaller NetCDF files that contained data for one day. The size of the resulting NetCDF files were about 640 megabytes (MB). Looking at the date from the current DST, the NetCDF file with the same date was loaded into the project containing the necessary temperature and depth values from OGCM.

Running the new solution gave an improvement of 89.33% in the run time. Instead of calling an external Python script, the OGCMs were loaded directly into the C# project. This meant it no longer took 100 ms every time any of simulations tried to retrieve a temperature value. However, this led to every simulation having its own copy of the OGCM which led to the system running out of memory. Therefore, this was changed so that all the simulations shared one version of the OGCM. This caused problems when simulations tried to read data from OGCM at the same time, which led to simulations getting the wrong values. To solve this a lock was implemented. The lock made sure that only one simulation was reading the OGCM at a time.

The number of trajectories generated by the simulation application increased noticeably as well. With these improvements running the simulation application with 200 simulations and a margin of error on temperature set to 1.0 °C gave two complete trajectories. To get two trajectories before these changes, using the same settings, required the simulation application to be run with 800 simulations. It is likely that

some of the returned values from the external calls to python were overwritten and that could be the reason the results improved.

Dividing the model into smaller 640 MB files helped, but the files were still much larger than expected. The original file was 48 GB and the smaller files that were generated totalled almost 40 GB. However, the new files only cover 63 days while the original file contained data for 274 days. It turned out that the smaller files did not compress the data like the original file. The data was stored as float64, which takes up a significant amount of space compared to using int16 which is what the original file did. Doing this reduced the size from 640 MB to 160 MB. Because of this compression any data retrieved from the OGCM must be converted using a formula. In Python it is done automatically, but not with SDSLite. This is the formula:

$$\text{Original Temperature} = \text{Compressed Temperature} \times \text{scale\_factor} + \text{add\_offset}$$

When the temperature value is extracted from the OGCM the "Compressed Temperature" is returned and then it is put into the formula. Both the scale_factor and add_offset are set when the NetCDF file is created and the files created in this project used the same scale_factor and add_offset as the original files provided by the IMR. The values are determined using these two formulas when the values are unsigned:

$$\text{scale\_factor} = \frac{\text{dataMax - dataMin)}}{(2^n - 1)}$$

$$\text{add\_offset} = \text{dataMin} + (2^n - 1) \times \text{scale\_factor}$$

DataMin and dataMax are the minimum and maximum values that any of the temperature values have and the $n$ is the number of bits in the packed data type which is 16 since int16 is used [45].

### 4.5.2 Parallel Computing

The simulation application started off as a program that only ran on a single CPU core and it used a considerable amount of time to be completed. Therefore, one of the supervisors suggested that the program should be run in parallel. This means executing the program on multiple CPU cores. To achieve this a parallel for loop [46] was implemented in the program and it led to a decrease in the run time.

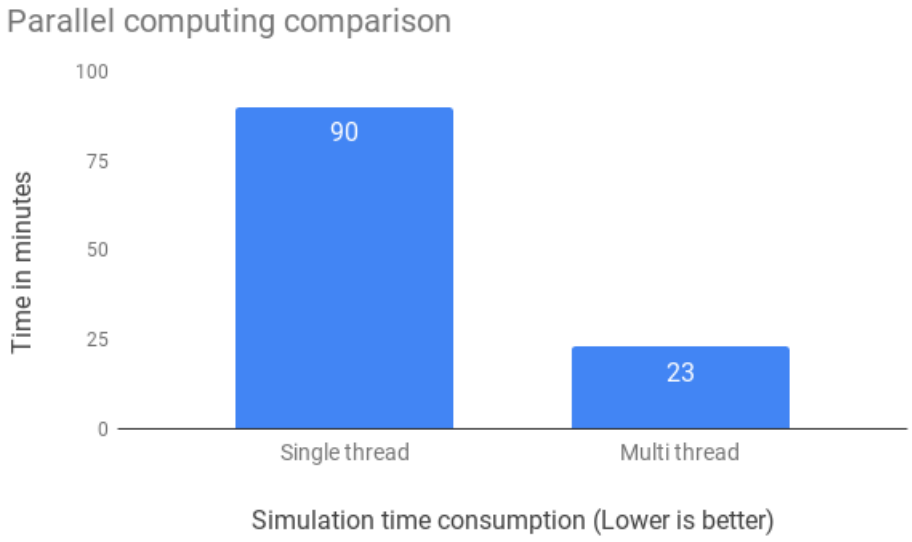



Figure 4.10: Illustration of the time reduction over time

Figure 4.10 shows how long the second iteration takes in seconds with and without running in parallel. Running the program in parallel lead to a 74.4% decrease in the run time, but it introduced new challenges. Variables and data sets that are shared across all the threads were constantly being overwritten. Threads that tried to get a value from an array would get the wrong one back because a different thread also tried to get it at the same time. Furthermore, the random variable that selects which valid location to choose was being overwritten. This led to the simulation application throwing an exception since not all the simulations had the same number of valid locations. Therefore, a BlockingCollection [47] was used instead of a regular array, since it is thread safe. After implementing a BlockingCollection in the simulation application, no data in those collections was overwritten.

Data extracted from NetCDF files could not be easily converted into BlockingCollections and had to remain as arrays. Therefore, a Lock [48] was put around code that called any function that retrieved data from an array. This was done to prevent threads from overwriting other threads when both were trying to retrieve data from the same array at the same time.

To implement the random selection of one of the valid locations a simple random variable was implemented. The variable could be any value between zero and the number of valid locations. This variable was being overwritten by other threads and caused the simulation to crash about halfway through. To solve this issue a static class called ThreadSafeRandom was created that would generate a random variable between 0 and the total number of valid locations.

## 4.6 System Recommendations

After running the simulation on multiple systems, it was clear that the system requirements depended on where the OGCMs are stored. If an HDD is used, a powerful processor is not required because the system is not able to retrieve the data fast enough. Therefore, the system requirements are lower for systems that us an HDD. Only the processor requirement is affected since the number of files that go into the memory does not change. A system with the minimum requirements should be able to run the simulation application, but might slow down the computer to a degree that the user cannot do much until the simulation application is completed. With the recommended configuration the user will be able to run the simulation without noticeably slowing down the system. Furthermore, the recommended amount of storage is the amount of space the OGCMs for DST 742 and DST 1664 use.

| | Minimum | Recommended |
|---|---|---|
| Operating System | Windows 10 | Windows 10 |
| Processor | Quad-core with SMT | Hexa-core with SMT |
| Memory | 8 GB RAM | 16 GB RAM |
| Storage | SSD (180 GB) | SSD (180 GB) |

Table 4.3: Recommendations for executing the simulation application with an SSD

| | Minimum | Recommended |
|---|---|---|
| Operating System | Windows 10 | Windows 10 |
| Processor | Quad-core | Quad-core with SMT |
| Memory | 8 GB RAM | 16 GB RAM |
| Storage | HDD (180 GB) | HDD (180 GB) |

Table 4.4: Recommendations for executing the simulation application with an HDD

# Chapter 5

# Discussion & Conclusion

## 5.1 Discussion

This thesis has produced a framework consisting of a C# application and a Unity3D project that work together to simulate, and display trajectories generated from data within DSTs and OGCMs. One part of the discussion will look at the objective aspect of the solution. This implies the run time and the number of trajectories generated by the simulation application. Second part of the discussion rely on the feedback provided by the experts from the IMR who tested the solution.

### 5.1.1 Algorithms

This project has attempted to implement three algorithms of generating trajectories from the DST data. However, only two of them were successful and both have their strength and weaknesses. The general algorithm has a clear advantage in run time, but is only able to generate half the number of trajectories.

Results from testing DST 742 shows that the merge algorithm generates more trajectories than the general algorithm, but at twice the run time. Furthermore, running the merge algorithm on DST 1664 generates more than twice the number of trajectories compared to running the general algorithm, but at twice the run time.

Initially, the length of DST 742 was set to 65 cm when running the simulation application. This length was incorrect, as the actual length was 82 cm. When the mistake was discovered, the number of generated trajectories tripled for the general algorithm and doubled for the merge algorithm. This gives a good indication that the calculated speed for fish is correct.

### 5.1.2 Time Consumption of the Algorithms

One of the goals of this thesis was for the run time to be so low that the simulation was in real time. This means that the user could make changes in the parameters and instantly see the effect it had on the generated trajectories. However, the massive amount of data required for running the simulation application, made it difficult to reduce run time. Excluding unnecessary OGCM data from the NetCDF files did decrease the run time, but it was still not close to real time. Figure 5.1 shows the total run time of the final prototype of the simulation application, using either DST 1664 or DST 742. The time was recorded from running the application with 10 000 trajectories. Reducing the number of trajectories would also reduce run time. Although, that would mean increasing the margin of error on temperature as well. For example, running the general algorithm on DST 742 with 100 simulations and a 2°C margin of error on temperature gives a run time of less than 2 minutes and 6 trajectories. Examining the CPU and SSD utilisation while the simulation was running revealed that the CPU spent most of run time waiting for files. Furthermore, reducing the number of simulation to 1 and and increasing the margin of error on temperature to 100 ° did not change the utilisation at all. Also, the run time was unchanged. This shows that the framework is not able to run the simulation application in real time without further reducing the size of the NetCDF files containing OGCM data. Otherwise, faster storage would be required.

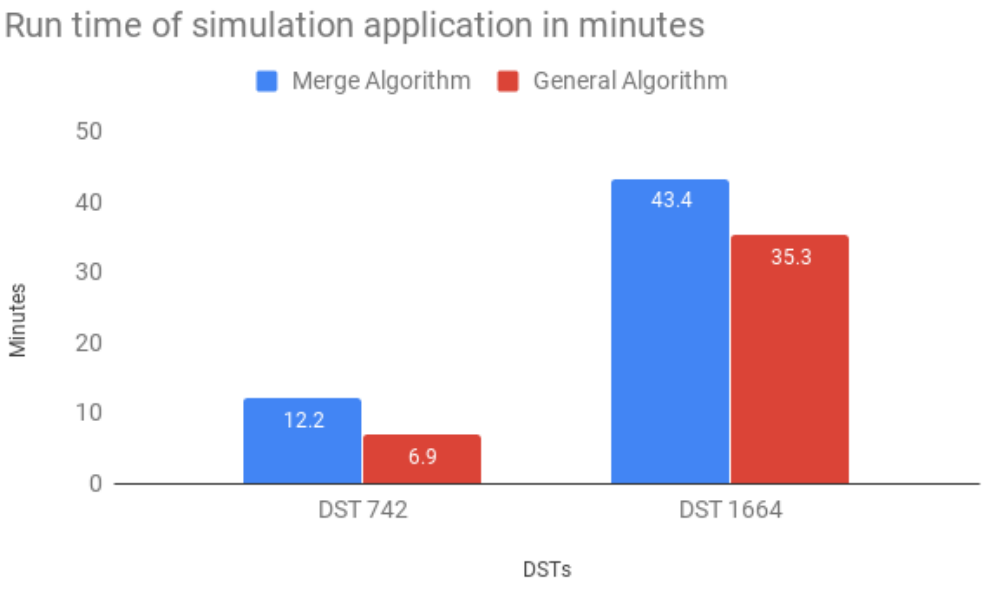



Figure 5.1: Comparison of run time from running the simulation application of DST 742 and DST 1664 using the General and Merge algorithms.

### 5.1.3 Visualisation Application

The prototype of the framework was tested twice, where feedback from the selection of experts mainly improved the usability and functionality of the visualisation application. The expert survey was the most important source of information since it revealed how much functionality the experts discovered and understood without help. Besides from studying the question sheet answered by the experts after completing the test (see Appendix A), an observation on how they interacted with the framework enlightened faults and errors. Testing on experts that have not been involved in the implementation of the framework revealed problems with the responsiveness of input fields. When testing, developers tend to click on the same place every time to make input fields react. By observing the experts click within the application, problems were discovered on when the input fields react to user interaction.

The results from the expert survey gave new ideas on functionality that improve the ability to analyse fish trajectories. The suggestions had to be discussed to determine if they were possible and then prioritised based on their importance.

**A1. Display date for each location in a fish trajectory**

**A2. Calculate average depth and temperature for locations within an area highlighted by the mouse**

**A3. Display several trajectories at the same time to see their similarities and differences**

**A4. Display the "best" trajectory, and sort trajectories from best to worst**

**A5. Ability to save trajectories and load them back onto the map**

All the suggestions except **A4**, were considered as possible to implement within the time span that remained. Suggestion **A4** was not considered. Defining one trajectory as better than other trajectories require guidelines that determine positive and negative characteristics of trajectories. The trajectories from a simulation execution are within the limits of the parameters given, and they have found a trajectory from release location to recapture location. All the trajectories are therefore possible solutions to where the fish has travelled, so they are all potentially the "best" trajectory. The remaining suggestions were discussed with the external supervisor

to determine which to prioritise for the implementation of the final prototype.

In agreement with the external supervisor, **A2** was removed as a potential functionality. Averaging values in a fish trajectory can remove scientific data since the resulting values can represent depths or temperatures that the fish would not prefer to be in. The remaining functionality is listed below in prioritised order.

**P1. Display date for each location in a fish trajectory**

**P2. Ability to save trajectories and load them back onto the map**

**P3. Display several trajectories at the same time to see their similarities and differences**

**P1** was prioritised since it was quick to implement, and it lets the user see where fish are at given dates. Experts on migration patterns have knowledge on where fish are during different seasons, e.g. located in Lofoten during spawning season. Seeing the date and the location of the fish can help them determine whether they believe that the trajectory display reasonable locations. **P2** gives the user the opportunity to save interesting trajectories or easily compare trajectories that have different parameter values. In the first prototype, all trajectories were replaced by the results from the latest execution of a simulation algorithm. This prevented the user from analysing trajectories with results from different simulations. **P3** was prioritised last due to its complexity and long implementation time compared to **P1** and **P2**.

In the final prototype, **P3** partially works. It displays as many trajectories as the user wants but deciding which trajectory the locations belong to is difficult, see figure 5.2. This is mainly because changing colour on each trajectory was harder than expected, so all trajectories are drawn with the same colour in the Unity3D scene. Figure 5.2 display the differences between displaying one trajectory (left picture) and two trajectories (right picture). When looking at the picture on the right, it is impossible to determine which trajectory each location belongs to. Improvements for this functionality would have been to colour code trajectories to see the differences between them, as well as being able to toggle which of the chosen trajectories to display at the same time.

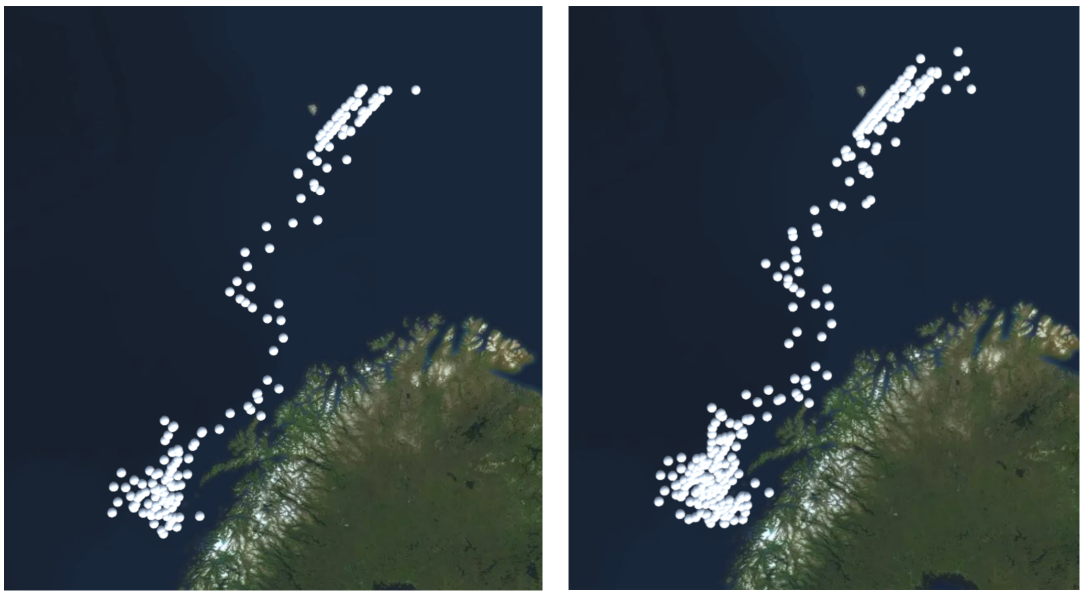


Figure 5.2: Display of one trajectory versus display of two trajectories

### 5.1.4 Simulation Application

The expert survey also resulted in a few suggestions on how to improve the simulation application of the framework. One of the experts suggested that always choosing the locations with the temperature closest to the temperature from the DST might give better results. To validate if it would be better, the simulation application has been run with this approach three times for each algorithm and the same has been done with the current approach. Furthermore, this was done to the DST 742 and DST 1664. Figure 5.3 and 5.4 show the average number of trajectories generated from all the simulations.

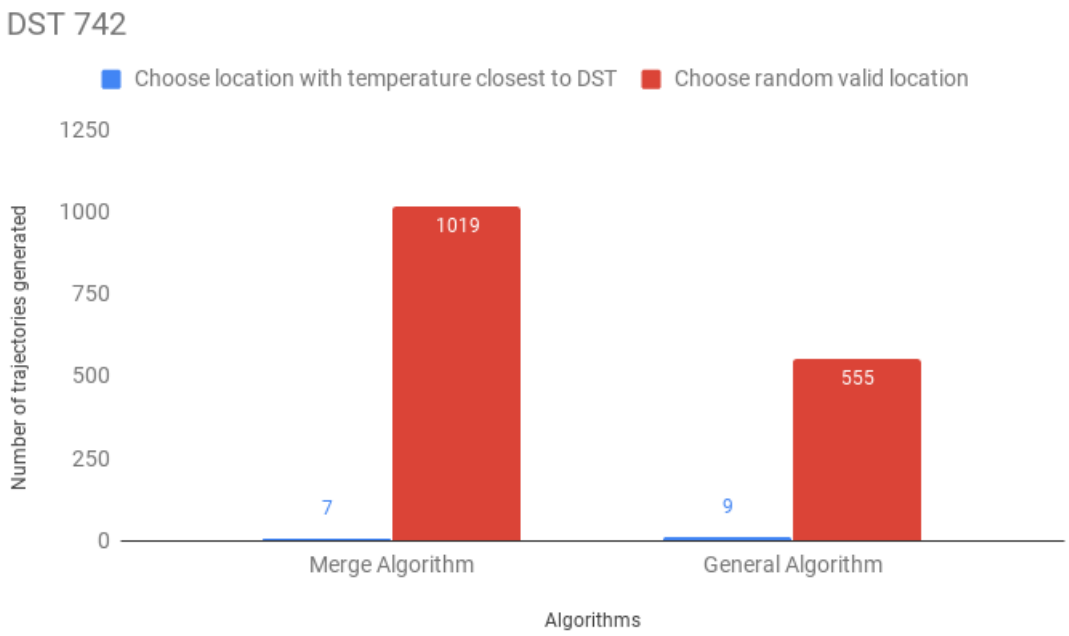



Figure 5.3: Comparison of how many trajectories are generated from the DST 742 when selecting a random valid location and the location with the temperature closest to the temperature in the DST.

Figure 5.3 shows that DST 742 can barely generate any trajectories regardless of which algorithm was used when selecting the next location with the temperature closest to the DST. Furthermore, the trajectories that were generated with that implementation travelled far past Lofoten where the recapture location is. Whereas, the merge algorithm generated trajectories that did not travel past Lofoten. However, it was still unable to generate the same number of trajectories generated when choosing the next location randomly and applying the weighting.

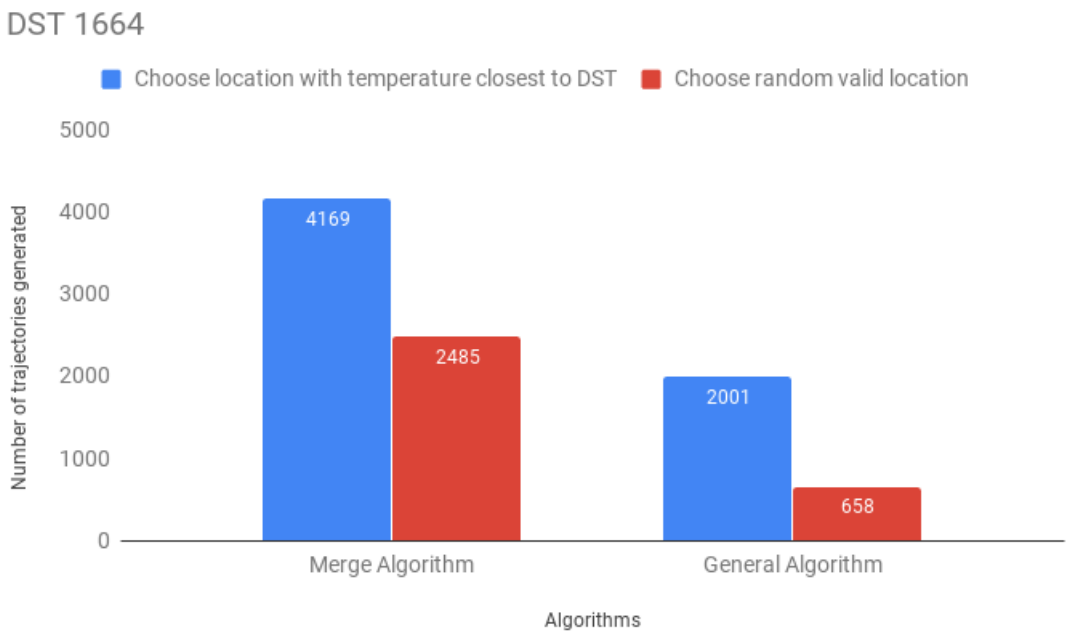



Figure 5.4: Comparison of how many trajectories are generated from the DST 1664 when selecting a random valid location and the location with the temperature closest to the temperature in the DST.

Testing this approach with the DST 1664 lead to significantly improved results, shown in figure 5.4. By choosing location with temperature closest to DST the merge algorithm increased the number of trajectories it generates with almost 1.7 times. Furthermore, the general algorithm saw an even larger increase of 3 times the trajectories. There is no weighting applied to DST 1664 unlike DST 742, which might be the reason why the results improved with this approach. Furthermore, the trajectories that were generated were not noticeable different with the new approach. However, since the number of trajectories increased, the margin of error on temperature and the number of simulations can be reduced.

It was also suggested by one of the experts that the simulation application should select the vertical z-coordinate with the temperature closest to the temperature in the DST data. This is because the measured depth in the DST data could have a margin of error of 30 meters. In order to test this approach, the temperature is found by looking at depth data from DST and OGCMs. Going through each vertical z-value in a (x,y) grid point in the OGCM, we select the temperature values from (x,y,z) grid points that has depth within 30 meters of the depth from the DST. For example, if the DST data has a depth of 100 meters, the simulation would store the temperature for all (x,y,z) grid points with depth between 70 - 130 meters. The temperature values are then compared to the temperature from the DST data, and the closest value is chosen. The idea is that more trajectories should be able to complete since it has several temperatures to choose from. However, as seen in figure 5.5 this was not the case. Testing this approach and comparing the number of trajectories that were generated with the previous approach, shows a significant reduction in trajectories. Also, the trajectories did not have any new variations that had not been seen before, and the run time did not increase or decrease. Therefore, it was decided that this approach would not be used.

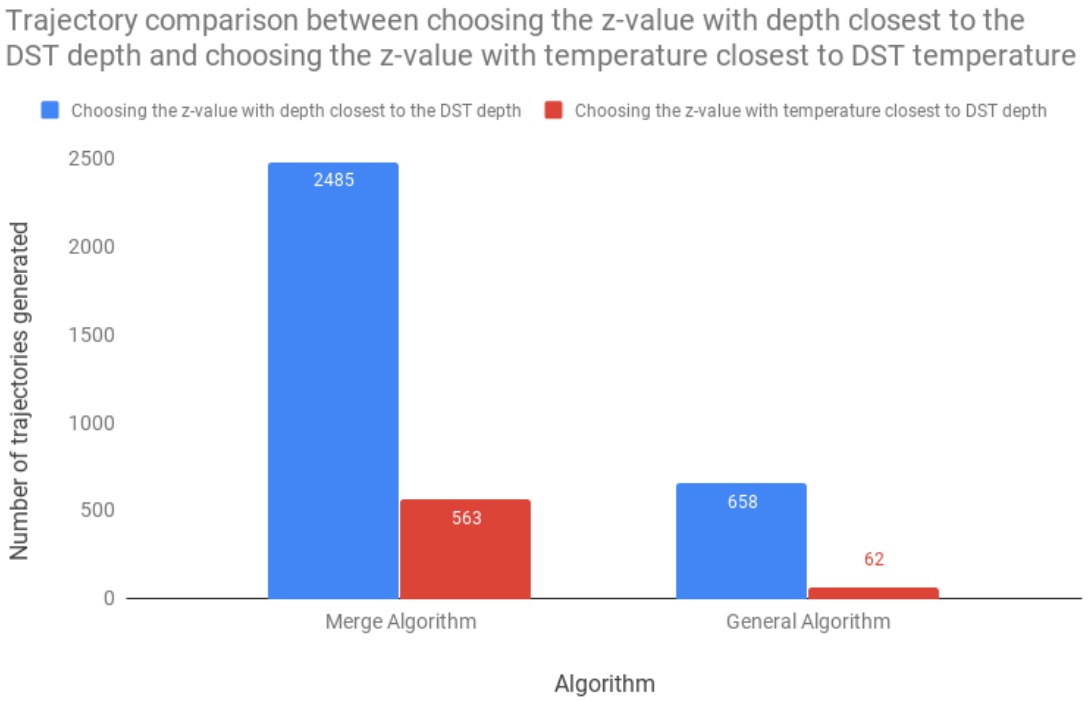



Figure 5.5: Comparison between choosing the z-value with depth closest to the DST depth and choosing the z-value with temperature closest to DST temperature

Figure 5.5 compares the results from running the simulation application using both the merge and general algorithm with and without choosing the temperature closest to DST temperature. Furthermore, the figure shows the average number of trajectories generated from multiple executions of the simulation application. This was done since there is some variation between each run because of the random speed.

Question 8 in the survey asks if the run time of the simulation application is too long. Two of the experts answered no, while the third one was not certain. They wanted the ability to visualise multiple trajectories at the same time. This makes sense since it is a well-known ecological problem that a median trajectory of all the trajectories would not be able to represent the movements of the individual animals [49].

When asked if they would use the software to simulate and visualise DST data in the future, two of the experts said yes and one said no. Two reason were given for saying no. First, this expert does not have easy access to a computer with Microsoft Windows installed. Second, the expert enjoys having full control over the analysis through own scripts since any graphical user interface can never provide the user with the same flexibility.

### 5.1.5 Switching Between Ocean General Circulation Models

Switching between OGCMs have led to an increased number of trajectories being produced compared to using only one of the available models. However, reviewing the results showed that few trajectories ever left the area covered by the Norwegian Sea model. Therefore, new conditions for switching to the Nordic Sea model were added to allow the trajectories to travel into the Barents Sea. This resulted in some trajectories within the Barents Sea. However, the number of completed trajectories were reduced and there were only a few trajectories that did cross into the Nordic Seas model. Also, the few trajectories that crossed into the Nordic Seas model did not travel far before returning to the area covered by the Norwegian Sea. Therefore, it is likely that the lower resolution of the Barents Sea model does not allow for low error margins on temperature and a low time step. Alternatively, it could be that the fish simply never did visit the Barents Sea.

## 5.2 Conclusion

The result from this thesis is a framework that transform data within OGCMs and DSTs to trajectories. The trajectories are visualised on a 2D map that contain analytic tools for modifying how trajectories are presented, and the information they display. The framework can run the simulation algorithms from the visualisation application and it makes sure to update the map when new trajectories are available. The thesis has four research questions that will be summarised in the sections below.

### 5.2.1 Simulation Application

**Q1. What parameters can be added to a solution that uses temperature and depth observations from DSTs and OGCMs to generate trajectories that are more realistic than the earlier solution?**

This thesis has presented a solution that generate trajectories using temperature and depth observations from DSTs and OGCMs. Several parameters have been implemented that were not presented in the earlier solution from [6]. The earlier solution only looks at surface temperature of the ocean and can generate trajectories that go through landmasses like islands. This solution looks at the temperature at the depth of the DST data and checks for land between the current location of a simulation and a potential new location before adding it to its trajectory. Additionally, this solution can simulate DSTs that cover more than two years with a time step of 1-day. The earlier solution [6] has a lower time step but does not show an example of a trajectory that covers more than 255 days. Ocean currents were also tested as a potential parameter for deciding where fish could travel by either having the simulation follow or go against the current. Unfortunately, it did not change the locations in the trajectories, and it increased the run time. Therefore, this parameter was removed from the solution. However, ocean current could still be used as a parameter in future solutions, but not the way it was implemented in this project. Additionally, dynamic weighting that weighted either towards or away from the recapture location depending on the time of year was tested on the DST 1664, but did not show any improvements. For that reason, it was not implemented. Moving on, whether the simulation should decide its next location based on weighting or the best temperature was also evaluated. It has shown improved results for DST 1664, but not DST 742. Therefore, it was implemented, but whether it should be used will depend on the DST used. Furthermore, this thesis also explored different ways of selecting temperature from the correct depth. First was looking at the z-coordinate closest to the depth of the DST. Second was selecting the z-coordinate with the temperature closest to the DST. Once both had been evaluated it was determined

that selecting the z-coordinate with the depth closest to DST depth was superior and was therefore implemented.

To answer **Q1**, this thesis has explored multiple new parameters that can be used when generating trajectories from DSTs and OGCMs. Such as how to select the temperature from depth layer and if the next location should be chosen at random or based on temperature. Also, an implementation of ocean currents was evaluated along with dynamic weighting. How these parameters affected the simulation application has been documented throughout the thesis.

### 5.2.2 Visualisation Application

**Q2. How should generated trajectories be visualised for scientists to find and analyse their ecological characteristics?**

The visualisation application is the controller of the framework. Allowing the user to visualise, analyse and generate trajectories, it aims to answer **Q2** in the best possible way. Starting with a 2D map and presenting it to the selection of experts located at the IMR, early feedback on their opinions was essential to visualise trajectories properly. Adding toggle buttons to change how trajectories are presented, allows the user to display trajectories as they want. Having more than one option on how to present them, makes it easier to satisfy all users. The user can either; display trajectories at once, display one-by-one location, draw lines between locations and only display release and recapture locations. Functionality that lets the user; choose which trajectory to display, run simulation algorithms and display information on trajectory locations, separates the application from an application that only visualise GPS locations on a map.

Feedback from question 12 in the expert survey (see Appendix A) was very positive. Two of the experts meant that the functionality improved their understanding of trajectories very much, while one of the experts meant that it did improve understanding.

To answer **Q2**, for scientists to find and analyse ecological characteristics, trajectories should be visualised by providing several options that modify how trajectories are presented on the 2D map. The user should also be able to extract necessary information, e.g. temperature, depth and date.

### 5.2.3 Performance

**Q3. How can generating trajectories be optimised to reduce the run time compared to the previous solution?**

The run time of the simulation has been recorded every time the simulation application has been run and reviewing the results shows that the goal of real time has not been achieved. Although the run time has been significantly reduced throughout the development of the simulation application, it is still not close to running in real time (see figure 5.1 in section 5.1.2). Even reducing the number of simulations to 1 could not get the run under 1 minute. This is because there are a lot of calculations being executed during the simulation application, but even if all the calculations took less than a second it would still not be in real time. This comes from the fact that the OGCMs are too large to be stored on the system memory and must be swapped back and forth to the storage every iteration of the simulation application. Therefore, running in real time is not possible using this solution with the hardware used in this thesis. It will require faster storage as well as greater computational power. This could be achieved by running the simulation application on a Graphics Processing Unit (GPU), but it would need enough memory to store all the OGCMs that the DST needs. DST 742 requires 30.8 GB and DST 1664 requires 141.2 GB.

Furthermore, the amount of data made it extremely time consuming to use latitude and longitude while generating trajectories like previous solutions have. Therefore, the x- and y- coordinates in the OGCM were used instead and showed a significant improvement in run time. This is likely the reason why two of three experts answered no to the question on the framework being too time consuming (see Appendix A). It appears that the run time will not be an issue for the scientist who would use this framework.

To answer **Q3**, this thesis has presented a solution with an improved run time compared to the previous solution [6]. This was done by approximating the location of the fish by using the x- and y-coordinates in the OGCMs, removing unnecessary data and limiting the directions the simulations check to eight predetermined directions.

### 5.2.4 Expert Opinions

**Q4. What functionality can be implemented into the framework to provide scientists with information on migration patterns and ecological characteristics?**

Through the project, feedback from the external supervisor and the selection of experts has been prioritised. The framework is created for their usage, so it is important to create a result close to their expectations. The spiral development model created for this project, intentionally had two phases for collecting feedback. Phase **II** and **V** assisted in how to present trajectories, what functionality to implement, and in what order to implement the suggested functionality.

Question 12 in the expert survey (see Appendix A) shows that two of three experts are *very satisfied* with the available functionality, while the last expert is *satisfied* with functionality. Even though the overall results were positive, the selection of experts suggested additional functionality to improve the framework, see section 5.1.3. Based on their feedback the final prototype was implemented to match their expectations.

To answer **Q4**, required functionality that needs to be included in this framework are collected from the experts. Their suggestions are summarised in the following; 1) the ability to run simulation algorithms from the visualisation application, 2) change how trajectories are displayed, 3) selecting which trajectory to display, 4) save trajectories and load them back onto the map, 5) display several trajectories at the same time, and 6) present information about data within trajectories. Combining this functionality into a framework, allow researchers to investigate migration patterns and ecological characteristics of fish.

# Chapter 6

# Further Work

## 6.1 Simulation Application

To improve trajectories generated from the simulation application, changes can be implemented to discover if they have any effect on the trajectory results. Dynamic weighting can provide better results on DSTs such as DST 1664, where release and recapture locations are close to each other. Weighting a trajectory towards known locations based on seasonal migration patterns, e.g. weight towards Lofoten during spawning season, can result in more accurate trajectories.

The attempt to use ocean currents in addition to the other parameters were not successful. An algorithm that could potentially limit areas the trajectories can traverse to by identifying turbulent ocean currents should be explored [50][51].

Brute force is used for finding locations in trajectories. Future work should focus more on analysing areas around locations, to decrease the run time of the framework.

## 6.2 Visualisation application

Presenting several trajectories at once is a functionality that can be further implemented. Altering the structure on how to draw several trajectories at once, can make it easier to draw them in separate colours. This would also make it easy to further implement that the user can toggle which of the chosen trajectories that should be visualised at the same time.

Expanding the visualisation application to a 3D world can provide deeper knowledge on migration patterns and behaviour within fish trajectories. Allowing the user to dive into the view of a fish would require recreating the topology, model the ocean with textures, and populating the ocean with species.

# Appendix A

# Expert Survey

**Question 1:**


**Question 2: Har du noen forslag til andre navn som gjør funksjonaliteten enklere å forstå? (Svarformat: gammelt navn - nytt navn)**

1. Show index of each position - Show each position consecutively
   Paths close to capture point - Paths ending close to capture point

2. Close to capture point, litt vanskelig å forstå, OK når det ble forklart

3. navnene var ok, men jeg foreslår at dere også oppgir dato for de ulike punktene, samt åpner for muligheten til å velge flere merker samtidig slik at man kan vise en poetsiell "vandringskanal"

**Question 3:**


**Question 4: Har du noen forslag til andre navn som gjør funksjonaliteten enklere å forstå? (Svarformat: gammelt navn - nytt navn)**

1. Distance = Distance between positions
2. Marker → Time step,
3. Navnene er ok, men foreslår at dere gir mulighet for å beregne gj.sn. verdier for et valg område/utsnitt av punkter

**Question 5:**

**Question 6: Har du noen forslag til andre navn som gjør parametrene enklere å forstå? (Svarformat: gammelt navn - nytt navn)**

1. -Weighting of path towards capture point (0.0 - 1.0): kanskje forklare at 0-0.5 er bort fra homing position, 0.5-1 er mot homing, og at 1.0 er det strengeste kravet.

   -Enheter på error on depth/temperature

2. Days ... tag data → Time step length in days,

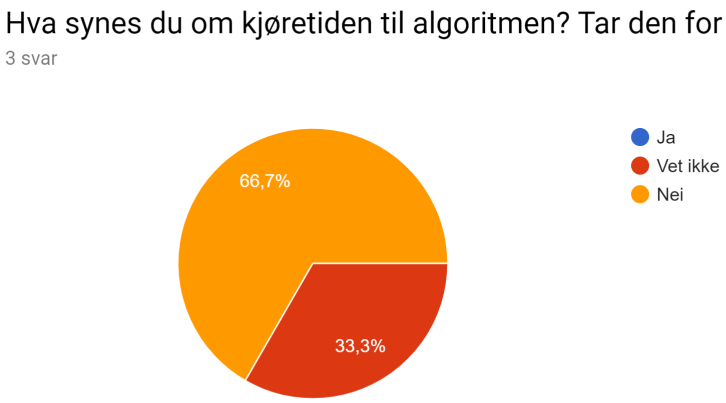
   Alternativt dersom holder på marker over → Days between markers

   Litt forvirrende at det heter marker over og tag data her

3. ser ok ut

**Question 7:**

**Question 8:**




**Question 9: Var det noe informasjon som manglet? (Hvis ja, spesifiser)**

1. -hvordan avslutte (trykke return)

   -Forklaring på at baren viser antall trajectoreier som når frem. Det er jo gøy informasjon.

2. Algorithm 0 er litt lite forklarende. Bedre å skrive ”forward” eller ”merge”

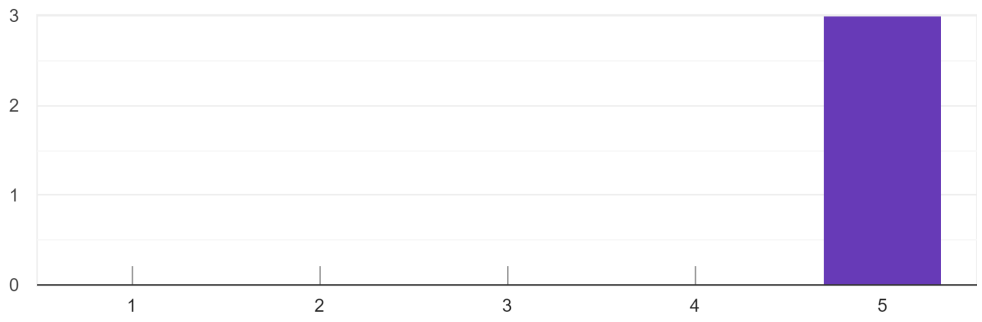
   Increment = Fiskelengde?, I så fall er fish length mer beskrivende

3. dato per punkt for å se hvor fisken er til ulike datoer, samt mulighet til å vise vandringsrute for flere merker samtidig

**Question 10:**

Hvordan synes du applikasjonen presenterer mulige ruter for fisk?

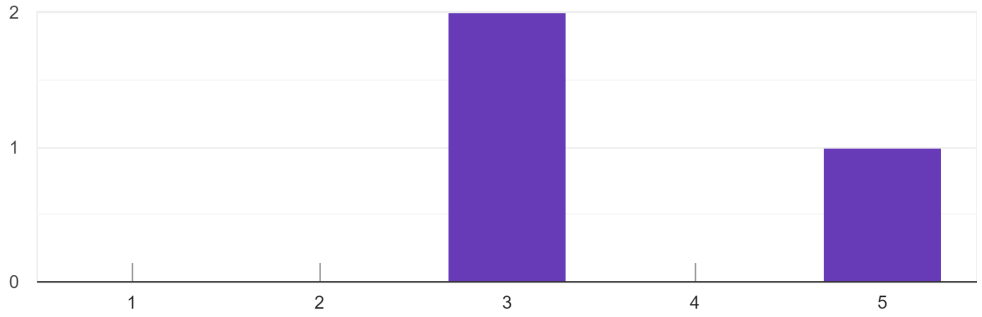
3 svar




### Question 11:

Med tanke på parametrene og fremgangsmåten som er brukt, hvor stor er sannsynligheten for at rutene kan være realistiske?

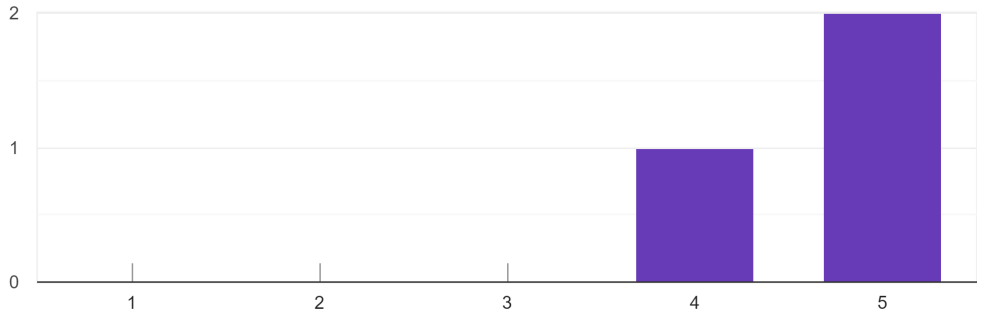
3 svar




### Question 12:

Bidrar de ulike funksjonalitetene i brukergrensesnittet til å få en bedre forståelse av rutene?

3 svar




**Question 13: Kommer du på noe funksjonalitet eller informasjon som mangler?**

1. - Valg av "beste" trajectorie, evt sortere treff etter dette

   - Kort info om algoritmen som er brukt

2. Save/Load knapper lagre simuleringene til en separat katalog med metadata om parametervalg, og kunne laste dette inn igjen uten å kjøre på nytt.

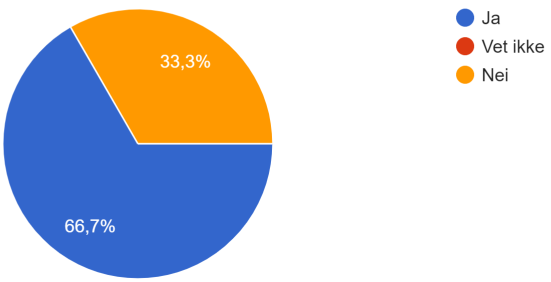
   Bare 8 retninger er begrensende på realismen. Likte ikke lengre trekk langs rett linje.

3. se tidligere svar

**Question 14:**

Ville du brukt et slikt verktøy for å få simulert og visualisert merkedata?

3 svar




**Question 15: Hvis nei, hvorfor ikke?**

1. To grunner, 1: Jeg har ikke lett tilgang til MS Windows,

   2: Jeg er litt nerd som liker å ha full kontroll på analysen via egne skript, en GUI kan aldri gi samme fleksibilitet til brukeren.

   Hadde vaert greit å bruke for å gjøre simuleringene, stiligere alternativ enn å redigere konfigurasjon av skript.

# Bibliography


[1] Svein Sundby. "Utviklingen innen oseanografisk forskning i Vestfjorden". In: *Fisken Hav.* (1980). URL: https://brage.bibsys.no/xmlui/bitstream/handle/11250/113125/fh_1980_01%5C%283%5C%29.pdf?sequence=1&isAllowed=y. (Accessed: 27.02.2019).

[2] D.H. Cushing. "Marine Ecology and Fisheries". In: *Cambridge University Press* (1975), pp. 99–106.

[3] Havforskningsinstituttet. "Årsmelding". In: *Fiskeridirektoratets Bibliotek* (1995).

[4] Havforskningsnytt. *Elektronisk merking avslører torskevandringa.* 1998. URL: https://brage.bibsys.no/xmlui/bitstream/handle/11250/115663/199810.pdf?sequence=1&isAllowed=y. (Accessed: 27.02.2019).

[5] N.C. Wells. "Reference Module in Earth Systems and Environmental Sciences," in: (2017).

[6] Bjørn Ådlandsvik, Geir Huse, and Kathrine Michalsen. "Introducing a method for extracting horizontal migration patterns from data storage tags". In: *Hydrobiologia* (2007).

[7] Kristin A. Cook and James J. Thomas. "Consistency in the behaviour types of the Atlantic cod: repeatability, timing of migration and geo-location". In: (May 2005).

[8] Ran Nathan et al. *A movement ecology paradigm for unifying organismal movement research.* 2008. URL: https://www.pnas.org/content/105/49/19052/. (Accessed: 12.04.2019).

[9] William J. Sutherland et al. *Identification of 100 fundamental ecological questions.* 2012. URL: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/1365-2745.12025. (Accessed: 13.04.2019).

[10] John Fieberg et al. *Regression modelling of correlated data in ecology: subject-specific and population averaged response patterns.* 2009. URL: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2664.2009.01692.x. (Accessed: 13.04.2019).

[11] Hacène Tamdrari et al. "The dispersal pattern and behaviour of Atlantic cod (Gadus morhua) in the northern Gulf of St. Lawrence: results from tagging experiments". In: (2012). DOI: 10.1139/F2011-137. (Accessed: 02.03.2019).

[12] Vilhjálmur Thorsteinsson et al. "Consistency in the behaviour types of the Atlantic cod: repeatability, timing of migration and geo-location". In: (2012). DOI: 10.3354/meps09852. (Accessed: 02.03.2019).

[13] Colin Ware. "Information Visualization: Perception for Design. 3rd ed." In: (2013).

[14] Barry Boehm. "Spiral Development: Experience, Principles, and Refinements". In: *Special Report CMU/SEI-2000- SR-008* (2000).

[15] Census of Marine Life. *Electronic tagging of marine animals*. 2017. URL: http://www.coml.org/comlfiles/scor/SCOR-tagging.pdf. (Accessed: 24.10.2018).

[16] Havforskningsinstituttet. *Merket torsk i Nord-Norge*. 2009. URL: https://www.imr.no/temasider/merkede_arter/merket_torsk/merket_torsk_Nord_Norge/nb-no. (Accessed: 17.09.2018).

[17] Institute of Marine Research. *Data-storage Tags*. 2004. URL: https://brage.bibsys.no/xmlui/bitstream/handle/11250/116126/No_2_Data-storage_Tags.pdf?sequence=2&isAllowed=y. (Accessed: 01.03.2019).

[18] Eva B. Thorstad et al. "The Use of Electronic Tags in Fish Research – An Overview of Fish Telemetry Methods". In: *Turkish Journal of Fisheries and Aquatic Sciences 13: 881-896* (2013). DOI: 10.4194/1303-2712-v13_5_13.

[19] Department of Oceanography Naval Postgraduate School. *Vertical Grid Points*. 2003. URL: http://www.oc.nps.edu/nom/modeling/vertical_grids.html. (Accessed: 24.10.2018).

[20] Aron Boone. *Vertical Hybrid-pressure Coordinate*. 2017. URL: http://aaron.boone.free.fr/aspdoc/node7.html. (Accessed: 18.03.2019).

[21] Titoxd. *Schematic of a sigma-z coordinate system*. 2011. URL: https://upload.wikimedia.org/wikipedia/commons/a/a1/Sigma-z-coordinates.svg. (Accessed: 01.05.2019).

[22] Vidar S. Lien et al. "Evaluation of a Nordic Seas 4 km numerical ocean model hindcast archive (SVIM), 1960-2011". In: *Fisken og Havet* (2013).

[23] Unidata. *Network Common Data Form (NetCDF)*. n.d. URL: https://www.unidata.ucar.edu/software/netcdf/docs/index.html. (Accessed: 23.10.2018).

[24] Unidata. *The NetCDF Interface*. n.d. URL: https://www.unidata.ucar.edu/software/netcdf/docs/netcdf_introduction.html. (Accessed: 25.10.2018).

[25] Microsoft. *Overview of the .NET Framework*. 2017. URL: https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview. (Accessed: 23.10.2018).

[26] BBVAOPEN4U. *The best programming languages for each task*. 2016. URL: https://bbvaopen4u.com/en/actualidad/best-programming-languages-each-task. (Accessed: 25.10.2018).

[27] Microsoft. *An introduction to NuGet*. 2018. URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget. (Accessed: 24.10.2018).

[28] CodePlex Archive. *SDS*. 2018. URL: https://archive.codeplex.com/?p=sds. (Accessed: 25.10.2018).

[29] Slant. *What are the best 3D C# game engines?* 2018. URL: https://www.slant.co/topics/4195/~3d-c-game-engines. (Accessed: 25.10.2018).

[30] Dave Kuhlman. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. 2012. URL: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html. (Accessed: 24.10.2018).

[31] Python. *About*. n.d. URL: https://www.python.org/about/. (Accessed: 22.10.2018).

[32] Unidata. *NetCDF4 API Documentation*. n.d. URL: http://unidata.github.io/netcdf4-python/. (Accessed: 18.10.2018).

[33] NumPy. *About NumPy*. n.d. URL: http://www.numpy.org/. (Accessed: 18.10.2018).

[34] Pythonconquerstheuniverse. *Static vs. dynamic typing of programming languages*. 2009. URL: https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/. (Accessed: 29.10.2018).

[35] IBM. *Compiled versus interpreted languages*. 2010. URL: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zappldev/zappldev_85.htm. (Accessed: 29.10.2018).

[36] Unity. *Game engines — how do they work?* n.d. URL: https://unity3d.com/what-is-a-game-engine. (Accessed: 21.10.2018).

[37] Mozilla. *WebGL: 2D and 3D graphics for the web*. 2018. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. (Accessed: 17.10.2018).

[38] Aleksandr. *Documentation, Unity scripting languages and you*. 2014. URL: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/. (Accessed: 23.10.2018).

[39] Unity. *Build once, deploy anywhere*. n.d. URL: https://unity3d.com/unity/features/multiplatform. (Accessed: 23.10.2018).

[40] Unity. *Standard Assets - Asset Store*. 2018. URL: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351. (Accessed: 23.10.2018).

[41] Mapbox. *About Mapbox*. n.d. URL: https://www.mapbox.com/about/. (Accessed: 29.10.2018).

[42] M Curatolo and Luciaon Teresi. "The Virtual Aquarium: Simulations of Fish Swimming". In: (2015). URL: https://www.researchgate.net/publication/313835946_The_Virtual_Aquarium_Simulations_of_Fish_Swimming. (Accessed: 02.03.2019).

[43] Vassily Lyutsarev. *Scientific DataSet Lite*. 2017. URL: https://github.com/predictionmachines/sdslite. (Accessed: 09.15.2018).

[44] Olav Rune Godù and Kathrine Michalsen. *Migratory behaviour of north-east Arctic cod, studied by use of data storage tags*. 2000. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.539.8880&rep=rep1&type=pdf. (Accessed: 20.03.2019).

[45] Unidata. *Writing NetCDF Files: Best Practices*. 2019. URL: https://www.unidata.ucar.edu/software/netcdf/docs/BestPractices.html. (Accessed: 06.05.2019).

[46] Microsoft. *How to: Write a Simple Parallel.ForEach Loop*. 2018. URL: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-write-a-simple-parallel-foreach-loop. (Accessed: 29.09.2018).

[47] Microsoft. *BlockingCollection¡T¿ Class*. 2018. URL: https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.blockingcollection-1?view=netframework-4.7.2. (Accessed: 25.10.2018).

[48] Microsoft. *Lock Statement (C# Reference)*. 2018. URL: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/lock-statement. (Accessed: 25.10.2018).

[49] John Fieberg et al. *Regression modelling of correlated data in ecology: subject-specific and population averaged response patterns*. 2009. URL: https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2664.2009.01692.x. (Accessed: 10.04.2019).

[50] *Lagrangian Coherent Structures*. 2014. URL: http://georgehaller.com/reprints/annurev-fluid-010313-141322.pdf. (Accessed: 29.04.2019).

[51] C. Shawn Shadden, Francois Lekien, and Jerrold E. Marsden. "Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows". In: (2005). DOI: 10.1016/j.physd.2005.10.007.