NTNU

Norwegian University of
Science and Technology

TMR4345 Marine Datalab
Ship Routing

Sida Yu
Student Number: 520598

Department of Marine Technology

May 8, 2021

Contents

List of Figures

List of Tables

1 Introduction

In this project a path planning system of a ship will be implemented. It is based on the map of west-norway ocean. the ship resistance is calculated with hollenbach's method, the added resistance from wave will be estimated by STAwave-1 formulae and the wind resistance will be include.The weather information could be grabbed from Meteorological Institute from website. In this way the fuel consumption could be estimated. with all these information a Dijlstra's algorithm could be implemented to find out the optimal route respect to energy consumption.

For this report, It is structured with following way. The second section will discuss about the theory of resistance estimation and Dijlstra's algorithm. the third section will talk about the simulation enviorment and the programming enviroment. In the forth section, the code structure will be discussed, I will introduce the task of each scripts and there data flow. In the section five, simulation result will be posted and discussed separately. The final section is a conclusion of the entire project. All the source code will be attached in the appendices.

2 Theory

2.1 Fuel Consumption

For this project is to build a fuel consumption based path planning.So In this part I would like to introduce the method of Fuel consumption estimation. For simplify the problem,The following assumption is applied

- The ship will travel with a constant velocity
- The ship thrust force is equal to the Resistance Force

with these assumption we could estimate the fuel consumption with following equation.

$$W = \int_{startpoint}^{endpoint} R_{ship}\, ds \tag{1}$$

Where:

W : The estimated fuel consumption
R_{ship} : Ship Resistance Force
ds : Unit distance between grids

Through integral the ship resistance along the route, we could estimate the fuel consumption.

2.2 Ship Resistance

From the previous section, It is very important to estimate the ship resistance in this project. The ship resistance is affected by velocity, displacement and hull form. Usually ship resistance is divided into three parts:

$$R_T = R_{calm\_water} + R_{wave} + R_{wind} \tag{2}$$

Where:

R_{calm_water} : Ship Resistance in calm water
R_{wave} : Wave Resistance
R_{wind} : Wind Resistance

In this project the calm water resistance is estimated with hollenbach's method, which will be introduced in the next section. Also STAwave-1 method is used to estimate wave resistance, the wind resistance estimated method is also introduced here.

2.2.1 Hollenbach's Method

Hollenbach's model is an estimate of the power requirement of a ship, which focuses on the pre- diction of the resistance in calm water.Thorsen (2019). This method is based on regression analysis of 433 ship modelsdepends on the ship hull dimensions. This method is more accurate than other method(Holtrop and Guldhammer_Harvald), For its large

scale of dataset.
First step of this method is to calculate the Froudes number as the following equation

$$F_n = \frac{V}{\sqrt{g \cdot L_{fn}}} \tag{3}$$

Where:

F_n : Froudes number
V : Ship Velocity
g : Gravity
L_{fn} : Froudes Length

Froudes length L_{fn} is varies with different relationship between L_{OS}(length over surface) and L(length between perpendiculars).The vale is defined as following table.

	L_{fn}
$\mathrm{L}_{OS}/L < 1.0$	L_{OS}
$1.0 < L_{OS}/L < 1.1$	L+2 / 3 $\cdot (L_{OS} - L)$
$1.1 < L_{OS}/L$	1.0667 $\cdot L$

Table 1: Definition of Froudes length L_{fn}Thorsen (2019)

With the calculation of Froudes number, The resistance coefficient could be estimated.

$$\begin{aligned} C_{R\text{ Hollenbach}} =& C_{R,\text{ Standard}} \cdot C_{R,FnKrit} \cdot k_L \cdot \left(\frac{T}{B}\right)^{a1} \cdot \left(\frac{B}{L}\right)^{a2} \cdot \left(\frac{L_{OS}}{L_{wl}}\right)^{a3} \\ & \cdot \left(\frac{L_{wl}}{L}\right)^{a4} \cdot \left(\frac{D_p}{T_A}\right)^{a6} \cdot \left[1 - \frac{T_A - T_F}{L}\right]^{a5} \cdot (1 - N_{Rud})^{a7} \\ & \cdot (1 - N_{Brac})^{a8} \cdot (1 - N_{Boss})^{a9} \cdot (1 - N_{Thr})^{a10} \end{aligned} \tag{4}$$

$$\begin{aligned} C_{R,\text{ Standard}} =& b_{11} + b_{13} \cdot F_n + b_{13} \cdot F_n^2 + C_B \cdot \left(b_{21} + b_{22} \cdot F_n + b_{23} \cdot F_n^2\right) \\ & + C_B^2 \cdot \left(b_{31} + b_{32} \cdot F_n + b_{33} \cdot F_n^2\right) \end{aligned} \tag{5}$$

$$C_{R,FnKrit} = \max\left[1.0, \left(\frac{F_n}{F_{n,krit}}\right)^{c1}\right] \tag{6}$$

$$F_{n,krit} = d_1 + d_2 \cdot C_B + d_3 \cdot F_n^2 \tag{7}$$

$$k_L = e_1 \cdot L^{e_2} \tag{8}$$

where:

T_A : Draught at AP
T_F : Draught at FP
T : Average Draught
L : Ship Length
B : Ship Width
D_P : Propeller diameter

N_{Rud} : Number of rudders
N_{Brac} : Number of brackets
N_{Boss} :Number of bossings
N_{Thr} : Number of side thrusters

Also the a, b, d and e with subscripts are coefficients as following:

$$a = [-0.3382, 0.8086, -6.0258, -3.5632, 9.4405, 0.0146, 0, 0, 0, 0] \tag{9}$$

$$b = \begin{bmatrix} -0.57424 & 13.3893 & 90.5960, \\ 4.6614 & -39.721 & -351.483, \\ -1.14215 & -12.3296 & 459.254 \end{bmatrix} \tag{10}$$

$$d = [0.854, -1.228, 0.497] \tag{11}$$

$$e = [2.1701, -0.1602] \tag{12}$$

After the Resistance coeficient figured out. The Resistance could be estimated:

$$R_{calmwater} = \frac{\rho}{2} V^2 \cdot (C_{RHollenbach} \cdot S + C_R \cdot B \cdot T) \tag{13}$$

2.2.2 STAwave-1 Method

When including the wave resistance into this project, The STAwave-1 method could be applied.This equation only void while the wave comes inside the limit of 0 to $\pm 45°$ The equation is states as following.

$$R_{wave} = \frac{1}{16} \cdot \rho \cdot g \cdot H_{\frac{1}{3}}^2 \cdot B \cdot \sqrt{\frac{B}{L_{BWL}}} \tag{14}$$

Where:

ρ : The water density
$H_{\frac{1}{3}}$:the significant wave height
B :the breadth of the ship
L :the length of the ship
L_{BWL} : is defined as the distance from the bow to 95% of maximum breadth on the waterline in metersThorsen (2019) meters

For we could get the significant wave from the website of Meteorolodisk institutt, We could estimate the wave resistance within a grid.For the lack of the wave direction information, In this case I assume that the wave always comes against the ship direction.

2.2.3 Wind Resistance

For the wind resistance. I used the following equation as a estimation method

$$R_{wind} = 0.5 * \rho_{air} * A_{proj} * C_{air} * V_{rel}^2 \tag{15}$$

Where:

R_{wind} : wind resistance
ρ_{air} :Air density

A_{proj} : Projected area on the surge direction.
C_{air} : Wind resistance coefficient.
V_{rel} : Relative velocity between wind and ship

In this project we would to apply this equation to a grid map.To calculate the resistance with in one grid, We could zoom in one node, and analysis the force.

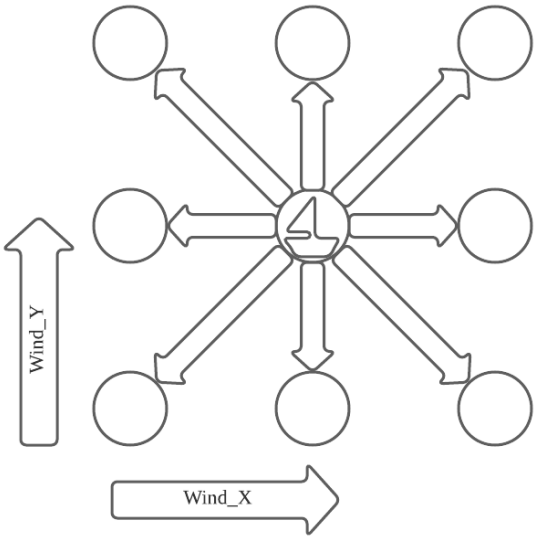


Figure 1: Relative velocity

From the plot we could find when a ship located in a node, there exist 8 possible motions:

$$M = (1,0),(0,1),(-1,0),(0,-1),(1,1),(-1,1),(-1,-1),(1,-1)$$

. So we could project the ship velocity to x,y directions. So the relative velocity could be describe as following:

$$V_{rel} = (\frac{V_{ship}}{norm(M)} * M - [V_{wind-x}, V_{wind-y}]) \tag{16}$$

For in this case we want to calculate the work of ship, So we only care about the relative velocity on the surge direction.

$$V_{surge} = norm(V_{rel} * abs(diag(M))) \tag{17}$$

In this way, we could calculate the value of the relative velocity on surge direction.And insert it into equation we could get the wind resistance

2.3 Dijkstra's algorithm

In this project Dijkstra's algorithm will be used to find the optimal route respect to power consumption. So in this section the theory of Dijkstra's algorithm will be introduced here. Dijkstra's algorithm is an algorithm for finding the shortest paths between the start node and destination,which is designed by computer scientist Edsger W.Dijkstra in 1956.Wikipedia (2021). The algorithm is states as following persude

1. First of all mark all the nodes as unvisited, which is created as a unvisited set
2. set the cost of start point as zero and all the other nodes as infinity.
3. move the current node to the neighbour nodes,which has not been visited. Compare the newly visited nodes cost to current cost and assign the smaller one to the cost.
4. when the node have been visited that node will not be considered in the neighbour motion
5. If the destination have been visited or all the nodes have been visited, The program will stop
6. else, repeat back to the third step

For better understanding I would like to share the pseudo code as following (from wikipedia)

```
1   function Dijkstra(Graph, source):
2       dist[source] = 0                                  // Initialization
3       create vertex priority queue Q
4
5       for each vertex v in Graph:
6           if v      source
7               dist[v] = INFINITY// Unknown distance from source to v
8               prev[v] = UNDEFINED// Predecessor of v
9
10          Q.add_with_priority(v, dist[v])
11
12      while Q is not empty: // The main loop
13          u     Q.extract_min()//Remove and return best vertex
14          for each neighbor v of u:// only v that are still in Q
15              alt      dist[u] + length(u, v)
16              if alt < dist[v]
17                  dist[v] = alt
18                  prev[v] = u
19                  Q.decrease_priority(v, alt)
20
21      return dist, prev
```

The code generated in this project is base on this pseudo code.

3 Approach

In this section I would like to discuss about how the theories applied into this project. First of all. The specified ship routing problem environment and the program environment will be describe

3.1 the ship routing problem

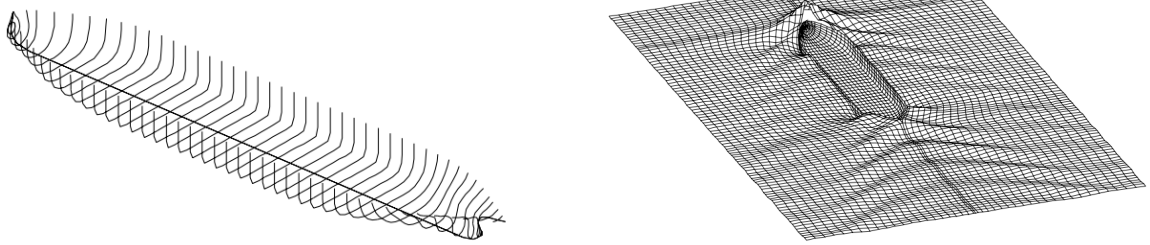
For simplify and specify the problem. I would like to solve the problem in a certain ship in certain day with in certain map.

3.1.1 Selected Ship

The hull form of a single screw ship has been chosen for this problem. For the resistance of this ship have been calculated in the work of Dunja MATULJALegovic & Dejhalla (2007) both in empirical method and the CFD method. So that it is easier to proof the validation through the comparison between my result and the result in this paper. The ship information is listed in the following table.

L_{pp}	169.00 m	L_{wl}	172.42 m
B	32.00 m	T	10.00 m
∇	42455.0 m^3	Δ	43506.0 t
C_B	0.785	LCB	+2.35 % L_{pp}

Table 2: Ship hull principal particulars



(a) Sections of the hull formLegovic & Dejhalla (2007)

(b) Ship Simulation plot in the literature.Legovic & Dejhalla (2007)

Figure 2: Ship plots from the work of Legovic.Legovic & Dejhalla (2007)

For verify the validation of our code of calculate the resistance in the method of Hollenbach. the comparison between CFD result from the literature and the estimated result from the code I use is shown as following.

	$R_T(kN)$			
V(kn)	POKUS OTPORA MODEL TEST	HOLTROP	HOLLENBACH (from literature)	HOLLENBACH (from code)
12	444.404	389.014	379.081	377.726
12.5	493.797	426.433	414.224	411.223
13	544.953	467.748	456.299	447.720
13.5	599.033	513.791	501.655	487.569
14	653.724	565.734	551.751	531.146
14.5	710.816	624.329	607.298	578.854
15	773.587	691.044	671.990	631.118
15.5	845.262	766.627	740.332	688.390
16	924.976	852.831	815.696	751.146
16.5	1020.841	951.385	898.977	819.887
17	1136.516	1063.929	996.875	895.140

Table 3: Compare the resistance between Literature Legovic & Dejhalla (2007) and the code calculation

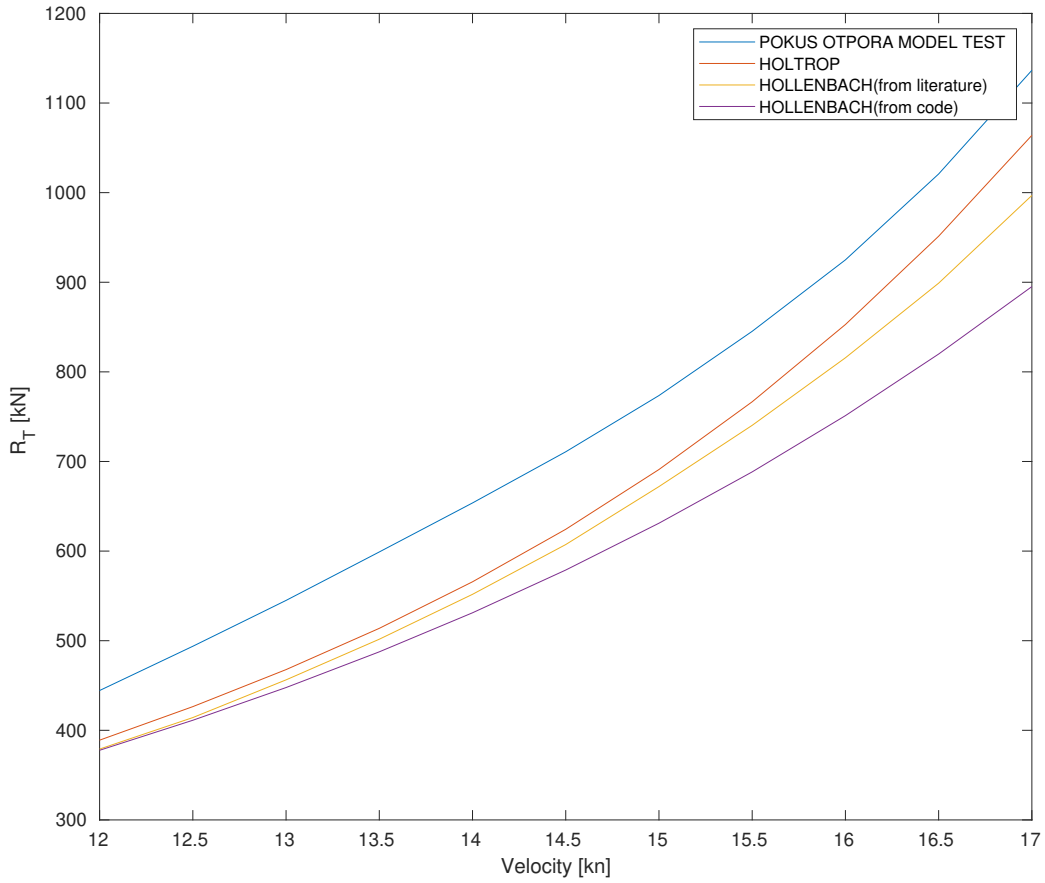
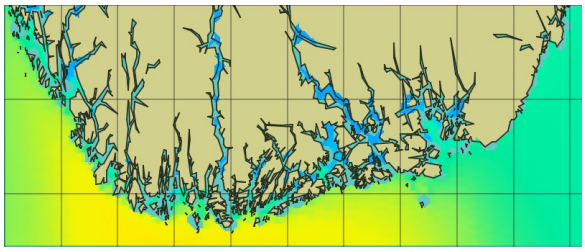


Figure 3: Compare the resistance between Literature Legovic & Dejhalla (2007) and the code calculation

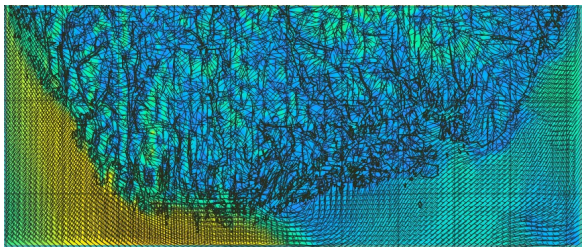
From the code we could find that the resistance I calculated with code is almost the same as the data from literature. In this way we could make sure the validation of my code. And all the parameters values are right.Also from the comparison between the

3.1.2 Selected Maps

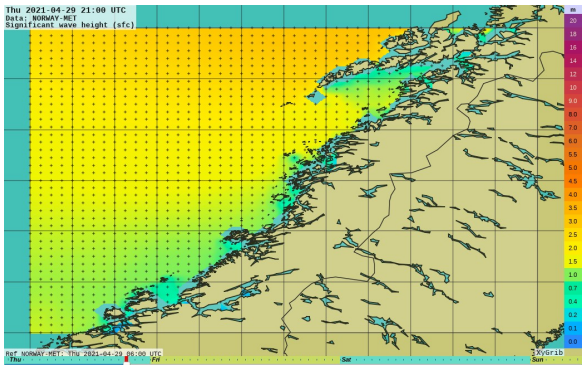
For the limitation of the weather information, The weather maps of West Norway,oslofjord,sorlandet,skag northsea,nordland,troms-finnmark are selected. For good look of the report I selected some of maps and post them as following.



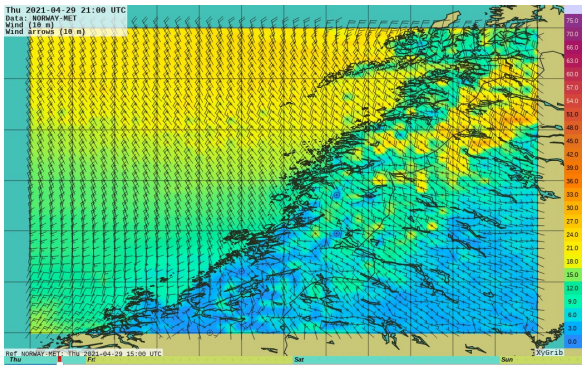
(a) Wave Data from West Norway



(b) Wind Data from West Norway



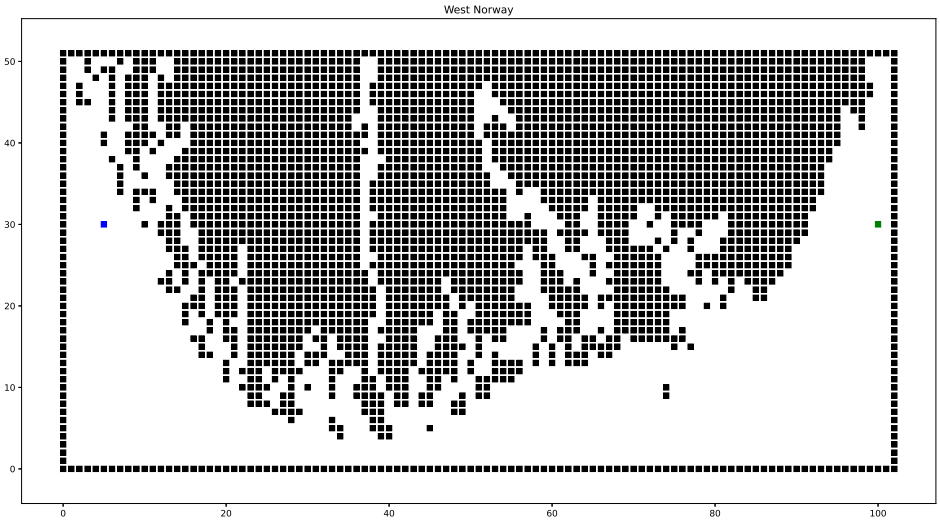
(c) Wave Data from Nordland



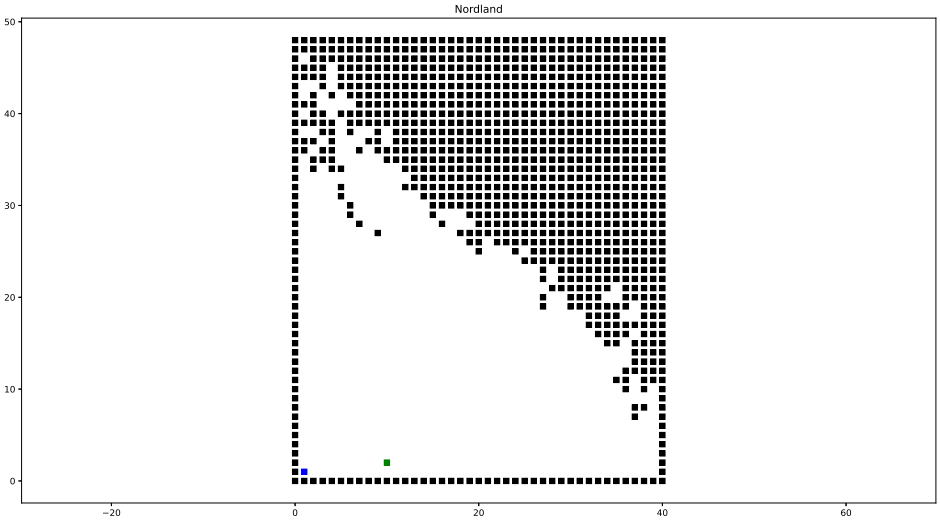
(d) Wind Data from Nordland

Figure 4: Weather information for West Norway and Nordland

in this way we could find there a part of land within the weather information area. So in this case it will be described as obstacle in this project. For visualization I plot the obstacle as black block in the plot,and it is shown as following.



(a) Obstacle shape of West Norway



(b) Obstacle shape of Nordland

Figure 5: Obstacle information for West Norway and Nordland maps

3.2 Programming in python

This work I would like to use python as the language of programming.Python is an interpreted high-level general-purpose programming language. Python is famous fro its object-oriented and functional programming. which will help me make the code more clear and easy to read. I have programmed some object class for the project, which will be introduce in the next section. Also Python is famous for the well-developed python packages. In this project I have used the following packages

3.2.1 List of packages

In this work, the python 3.6.9 64-bit environment is being used and the following packages are installed.And all of these packages will be used in my program.

Numpy:

Numpy is a package NumPy, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.Harris et al. (2020)In this project I would like to use this package to calculate the matrix.

Heapq:

Heapq-Heap queue algorithm provide an implementation of priority queue algorithm. Heaps are binary trees, in which every parent node has smaller or equal value than its children nodes. In this project, This modeule will be used in the search process of Dijkstra's algorithm to find out the min power cumptiom route

Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. In this project this package will be used to visualization of the process for the searching the optimal route

Pupygrid:

Pupygrib (pronounced puppy grib and short for PUre PYthon GRIB) is a light-weight pure Python GRIB reader. It provide a easy method to extract data from GRIB file. IN this project a weather dataset is being set up. With this package helps me to extract data and calculate the resistance force, which will be introduced in the next section.

PySimpleGUI:

PySimpleGUI is a wrapper to the Tkinter Python API that allows the programmer to utilize all the same UI elements as with Tkinter but with a more intuitive interface. PySimpleGUI was established in an effort to create a more user-friendly Python GUI development process.Everybodywiki (n.d.) With this package a GUI will be implemented to select the map and start point and goal point.

4 Code Structure

This chapter will explains how the program is being built. The introduction of each python file will be involved in. Its functions and a flowchart of the whole program is presented

4.1 The Python program

GUI.py
this file is the framework of this program. Here a user interface is implemented.The GUI window is designed as following plot.

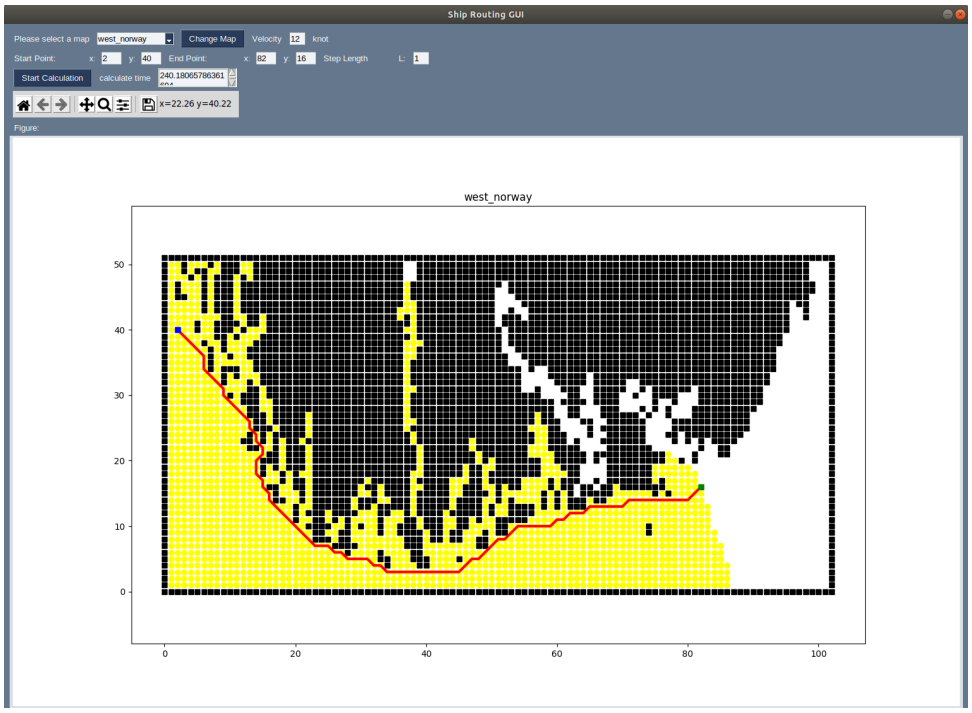


Figure 6: Overview of the GUI

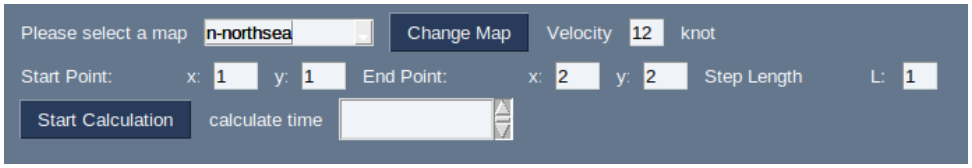


Figure 7: Control area of the GUI

The user could select map within the range of combo box, and its default value is 'West Norway'.Also you could set the ship velocity ,start point, end point and step length in the corresponding area. when everything settle down you could press the button 'Start Calculation' to start the program and after a while the result will show up. the waiting time depends on the map and dataset.The following function is included in this file

1. initial function: here is the layout of the GUI window and the main loop of the GUI
2. show_map: when we press the button of 'change map' the grid plot of the map will show up
3. cal_route : when the button 'Start Calculation' is pressed this loop will start and start searching the optimal route
4. draw_figure_w_toolbar : here is a function which could plot with a toolbar

Dijkstra.py
In this file, the Dijkstra class is defined and implemented, which contained a set of functions:

1. initial function : here the start point, end point, map name, step length and ship velocity will be imported. With the map name ,the wave and wind information will be imported into this program.
2. searching: here is the main loop of the Dijkstra's method,which is introduced in the section 2
3. get_neighbor: find all the neighbor nodes of the current node, which is not in the obstacles
4. cost: here is a function to calculate the power consumption of current node
5. is_collision: here is a function to detect whether a node is inside obstacles
6. out_of_range: here is a function to detect whether a node is out of the data range
7. extract_path: when the end point is in the set of visited,This function will help us to extract the optimal path

Resistance_Forces.py
A class of ship resistance is built here which contains 4 function:

1. Initial function will import the ship parameters and weather data address from file Ship_Model.yaml. According to the address, the wave and wind data will be imported from Folder weather_dataset.
2. hollenbach is a function of ship velocity,Which will calculate the clam water resistance force for the ship introduced in the section 3.
3. hollenbach_map function:For the clam water Resistance is Constant when the ship velocity is constant. To speed up the calculation, I use a map instead of a function to calculate.
4. Wind_resistance function: This is a function for calculate the wind resistance within a grid. With the weather information from weather dataset
5. Wave_resistance function: Here the wave resistance will be computed within a grid

Obstacles.py
In this file, A map of obstacles will be build up here.Two function is included here

1. initial function: here the program will import the wave information to build up the map

2. obs_map function: when there is no wave information for a node, There is the land, In this case it is set as the obstacles. Also the boundary of the data will be set as obstacles.

get_grib.py
Here is a portal for import the data from the weather dataset.

1. Get_wave_data: here is a function to get the wave significant height
2. Get_wind_data: here is a function to get the wind velocity which is projected on x and y direction.

plotting.py
Here is a class for visualize the optimal path for the program

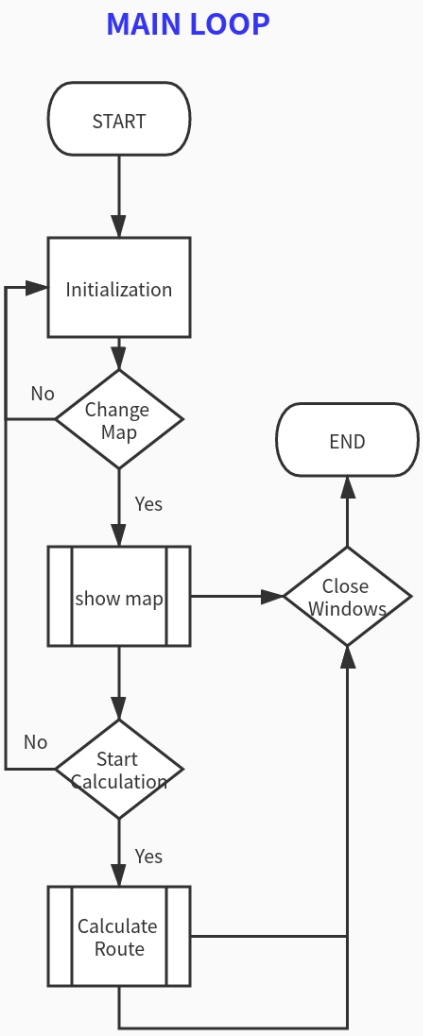
1. initial function: here the start point, end point and the map name will be imported
2. plot_grid: here is a plot function to plot the grid and obstacles maps
3. plot_visited_in_GUI:Here is a function to plot the visited nodes as a picture
4. plot_visited:Here is a function to plot the visited node as a animation
5. plot_path: Here is a function to plot the optimal path as a red line.and the start point as blue node , end point as green node.

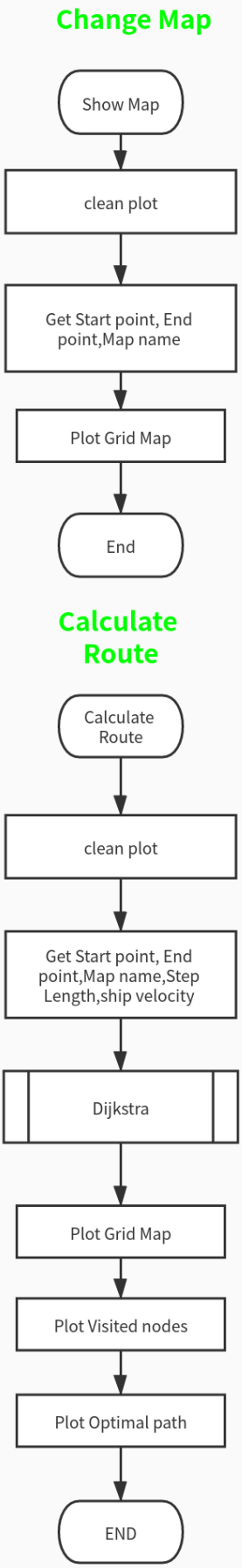
Ship_Model.yaml
Here is a file contains the ship information and the weather data address

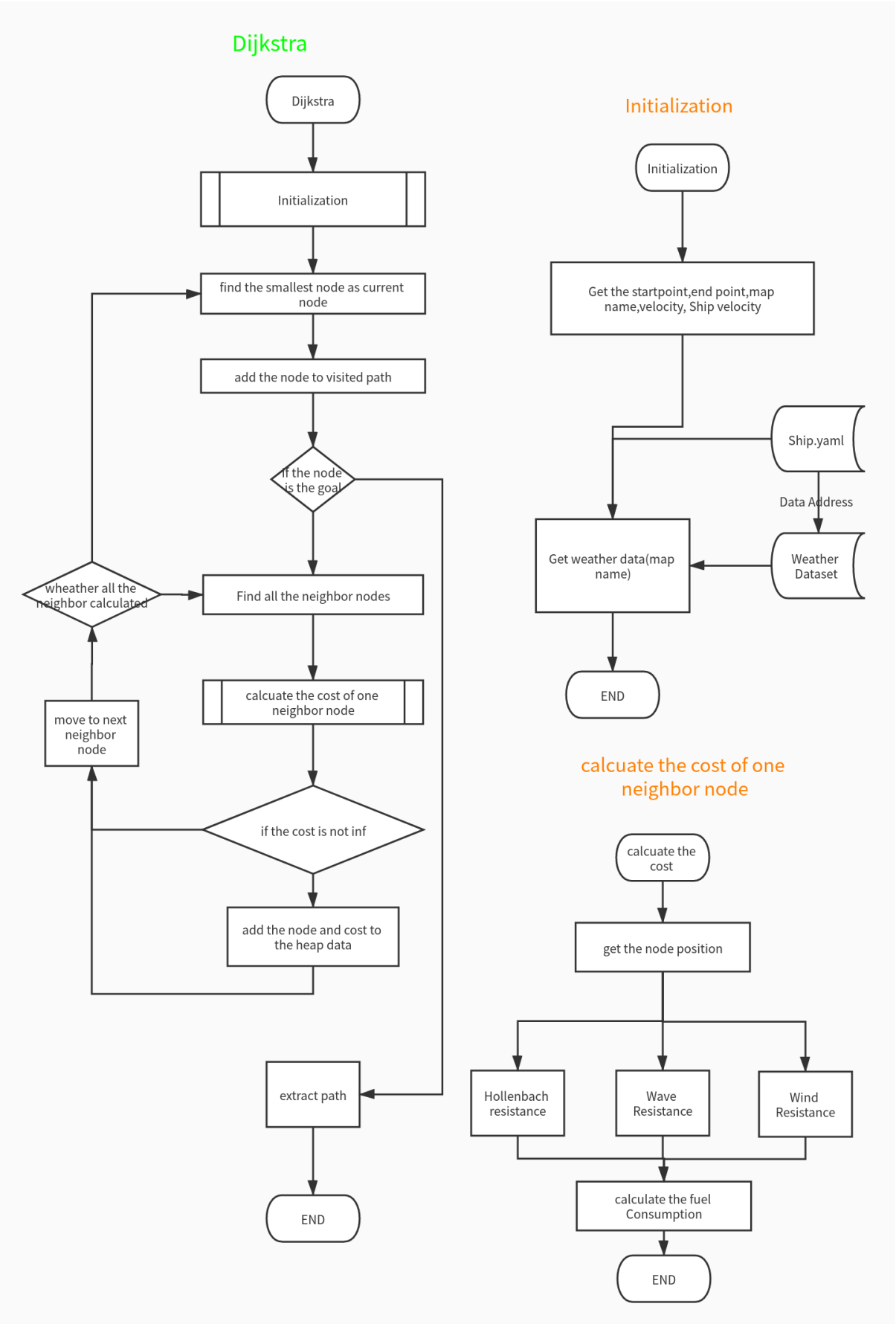
weather_dataset
Here is a dataset of the wave and wind for different maps

4.2 Flow Chart

For better understanding, The Flowcharts are included in the next two pages,For distinguishing, The main loop title is colored by blue,and its subroutines are colored by green. The subsubroutines are colored by orange.


MAIN LOOP
START
Initialization
No
Change Map
Yes
show map
Close Windows
END
No
Start Calculation
Yes
Calculate Route


Change Map
Show Map
clean plot
Get Start point, End point,Map name
Plot Grid Map
End
Calculate Route
Calculate Route
clean plot
Get Start point, End point,Map name,Step Length,ship velocity
Dijkstra
Plot Grid Map
Plot Visited nodes
Plot Optimal path
END


Dijkstra
Dijkstra
Initialization
find the smallest node as current node
add the node to visited path
If the node is the goal
Find all the neighbor nodes
wheather all the neighbor calculated
calcuate the cost of one neighbor node
move to next neighbor node
if the cost is not inf
add the node and cost to the heap data
extract path
END
Initialization
Initialization
Get the startpoint,end point,map name,velocity, Ship velocity
Ship.yaml
Data Address
Get weather data(map name)
Weather Dataset
END
calcuate the cost of one neighbor node
calcuate the cost
get the node position
Hollenbach resistance
Wave Resistance
Wind Resistance
calculate the fuel Consumption
END

5 Results

In this section I ran three set of experiment as Case study A, Case Study B and Case Study C In case study A, I would like to run the program in different maps which is available in the dataset,and compare the results. In case study B, In order to find the influence of ship velocity on the planned route results, I run the program on the map of West Norway with different ship velocity. In the case study C, I try to find the relationship between the step length with the calculation time . And the discussion will be followed with each group of result.

5.1 Case Study A

In this case I would like to try to plan the path with different maps. The start point and end point are set at the different side of the map to make the ship get through the most part of the map. And the results are shown as following plots

Ship velocity	12 knot
Step Length	1

Table 4: Simulate condition

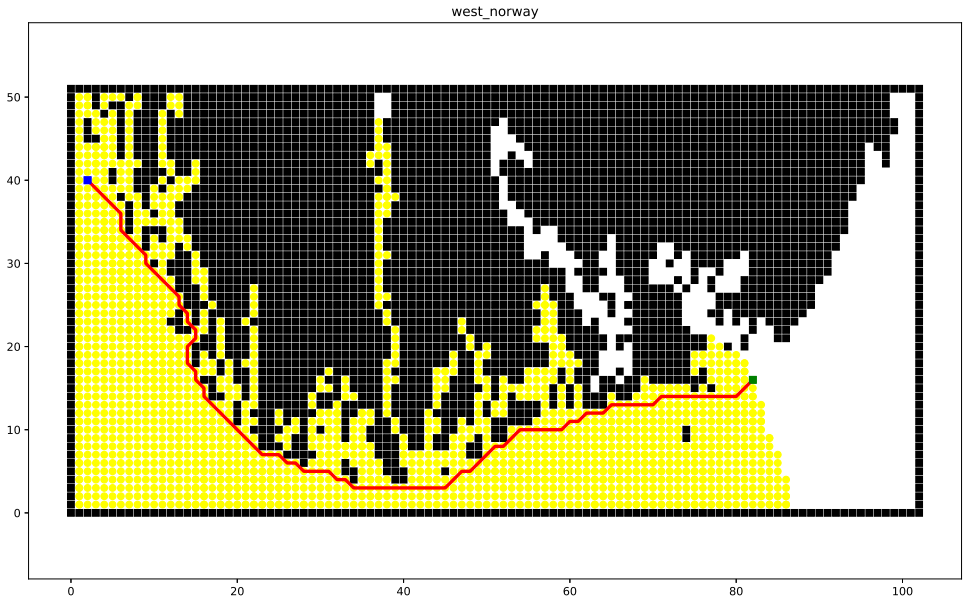


Figure 8: West Norway

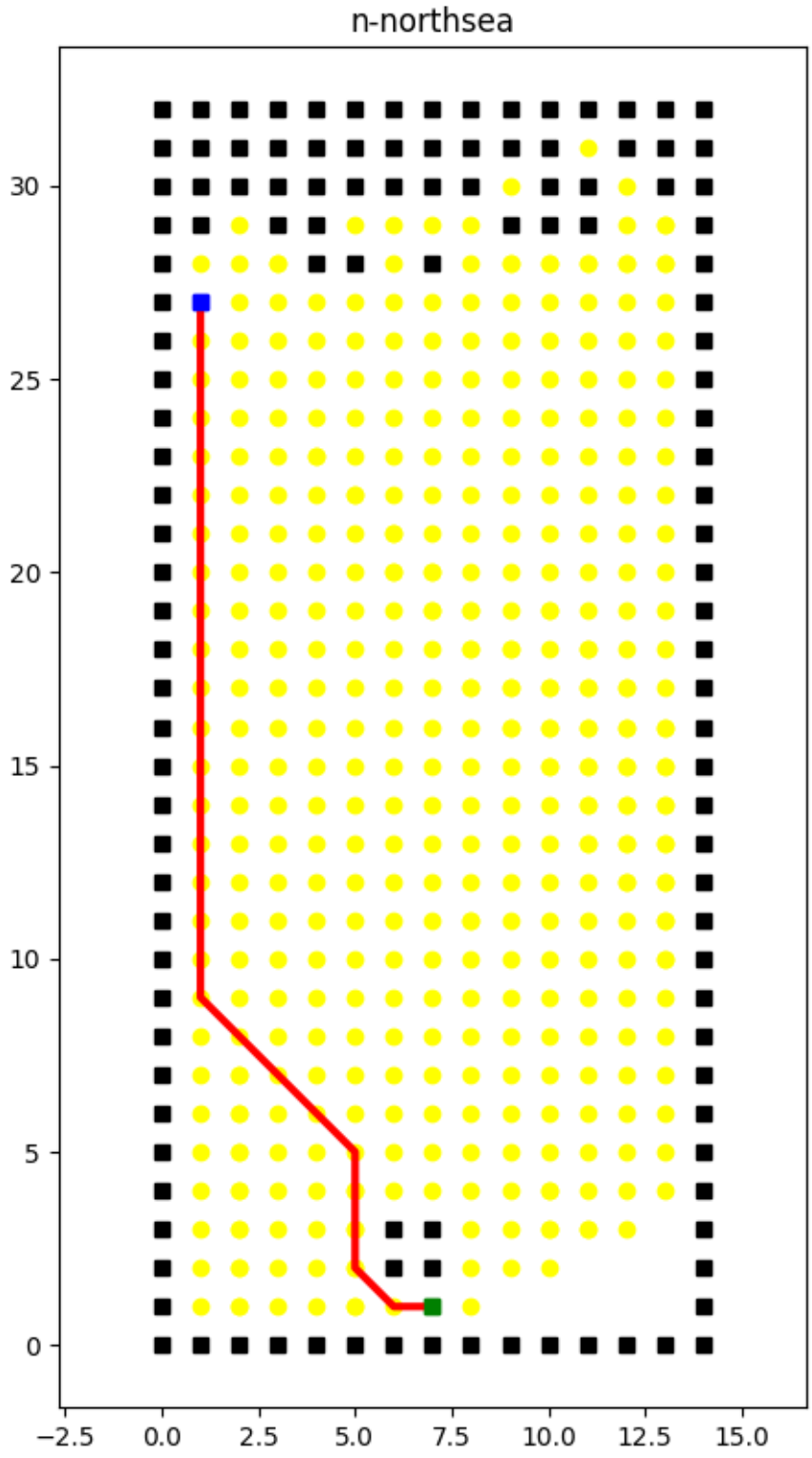


Figure 9: N-Northsea

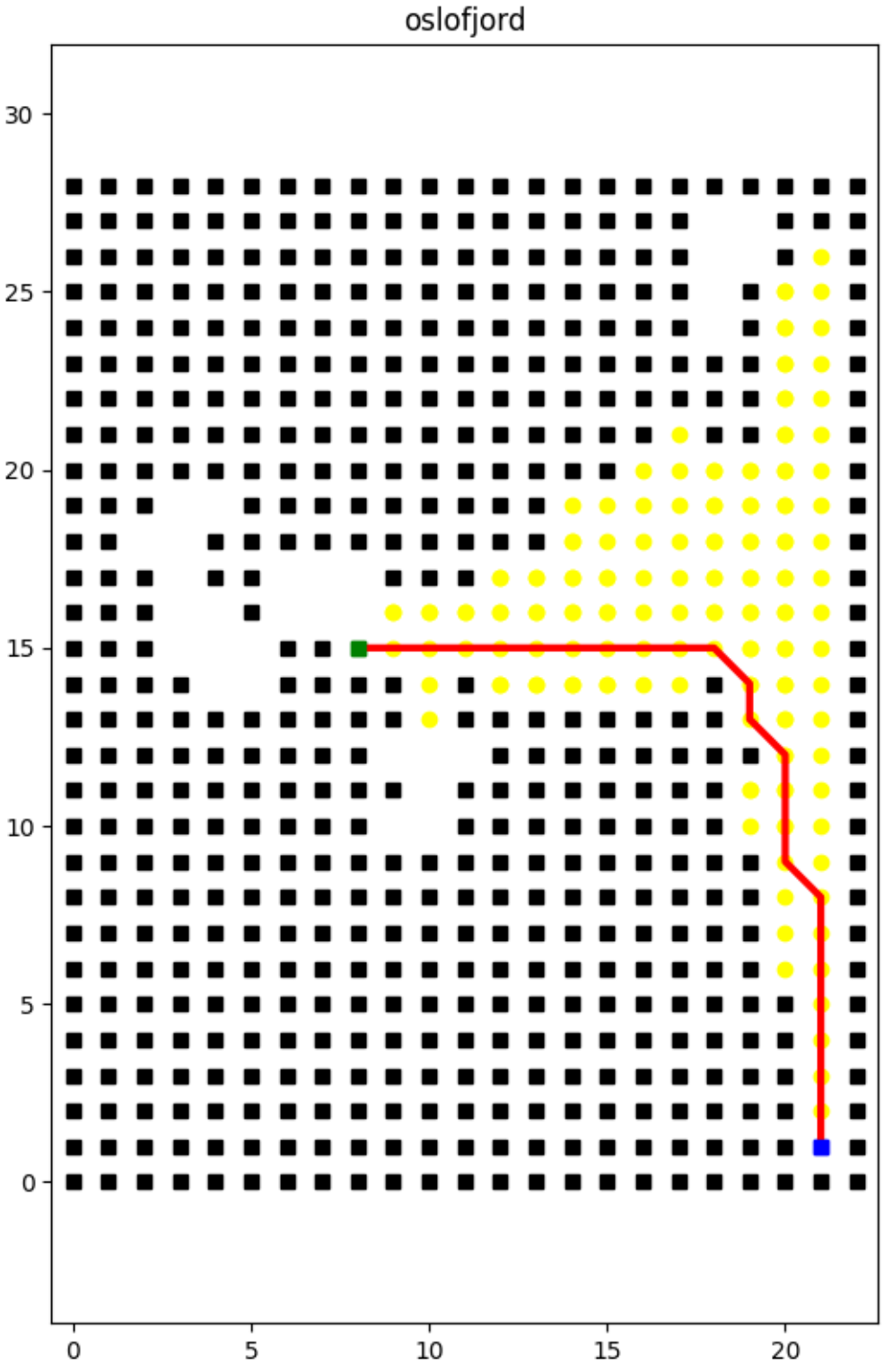


Figure 10: Oslofjord

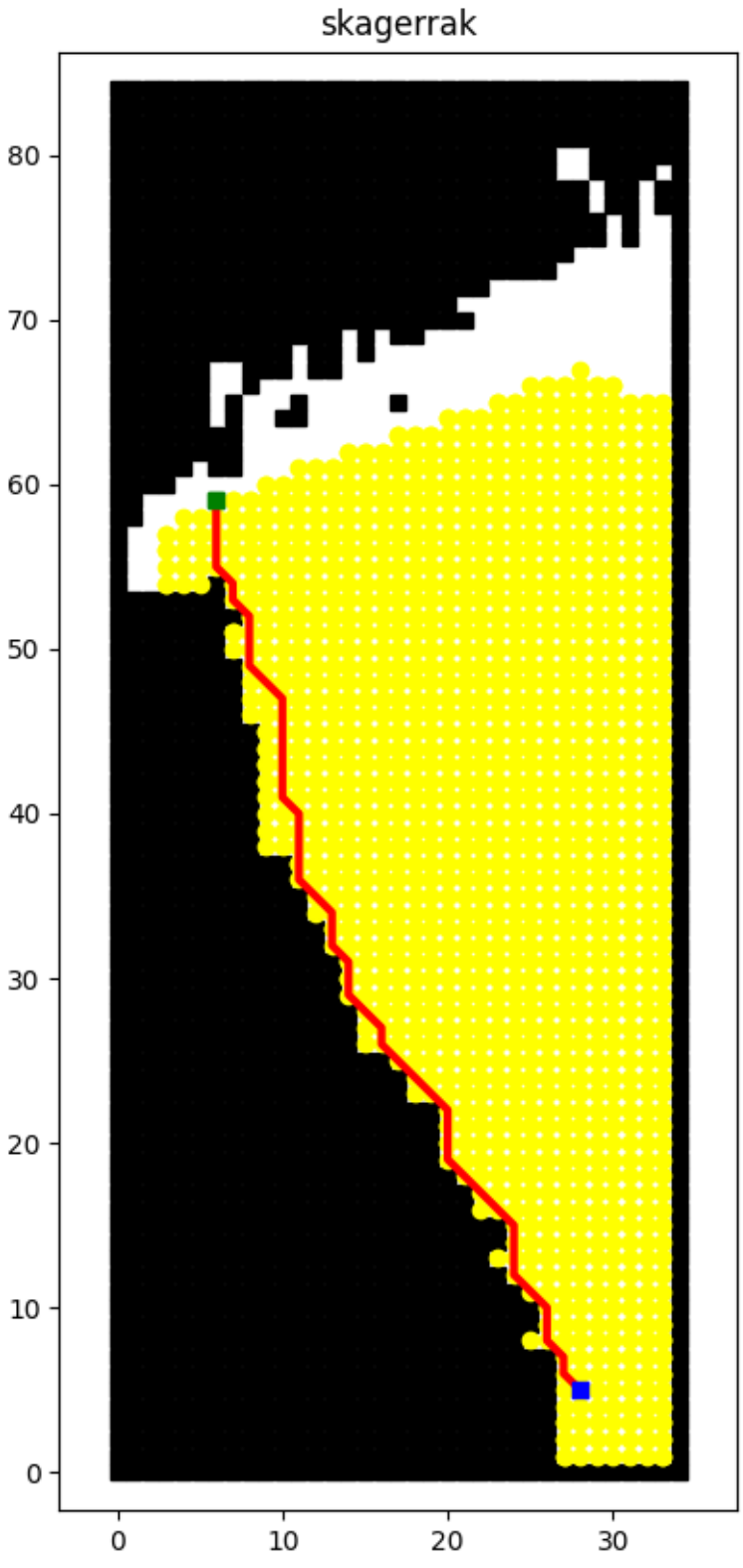


Figure 11: Skagerrak

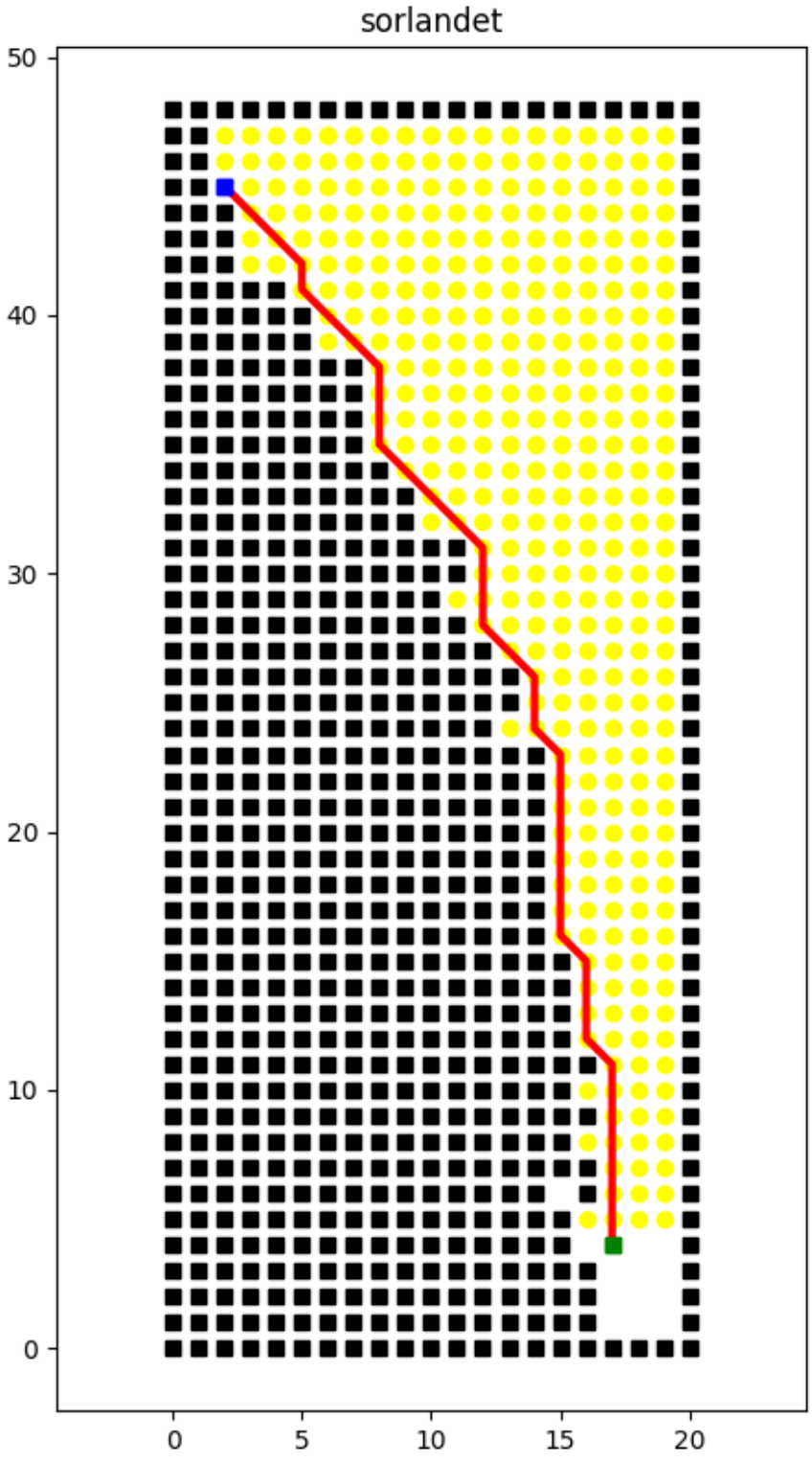


Figure 12: Sorlandet

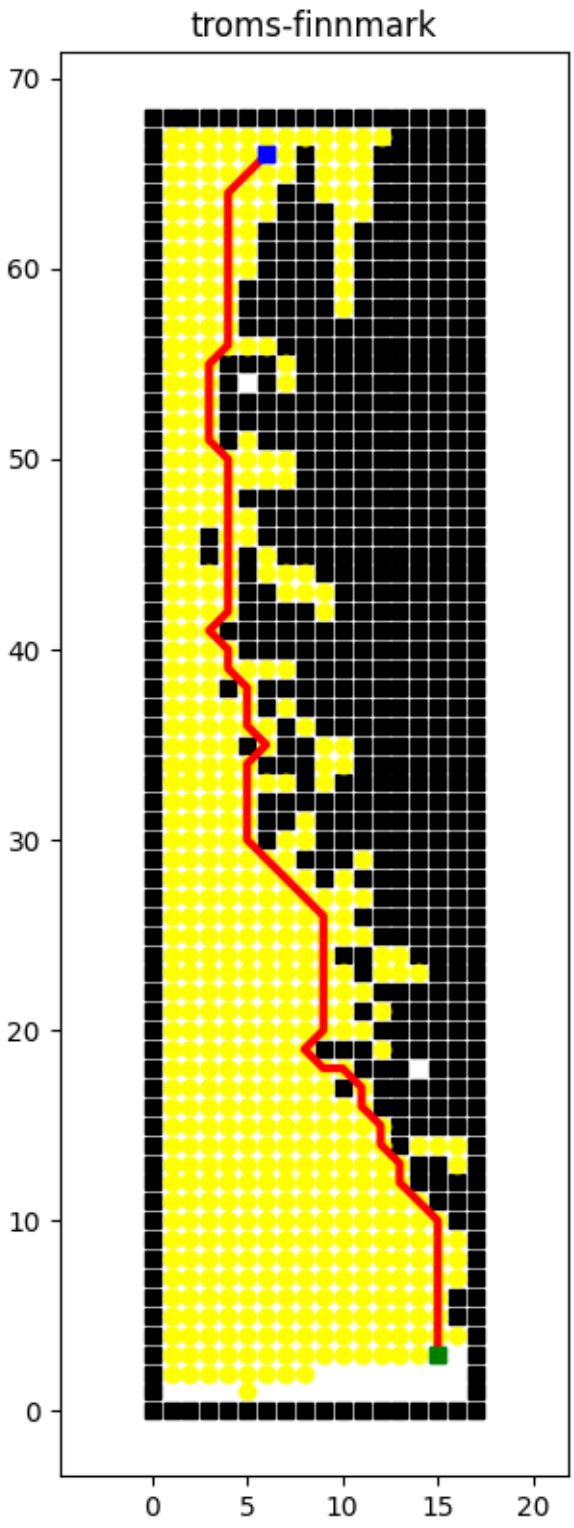


Figure 13: Troms-Finnmark

From the Result we could find that the ship tend to track the shape of land, For the wave

height and wind velocity are much smaller than the offshore parts. So in some case, It shows much more trend to travel along the land than straightly to the destination. This trend is more obervious in the map of West Norway,Skagerrak and Troms-Finnmark

5.2 Case Study B

In this case I would like to look into the influence of ship velocity on the routing process.So I test the ship travel from (40,2) to (92,22) in the map of West Norway with the speed of 12,17,22,27,32 knot separately. and compare the result with the shortest distance route.

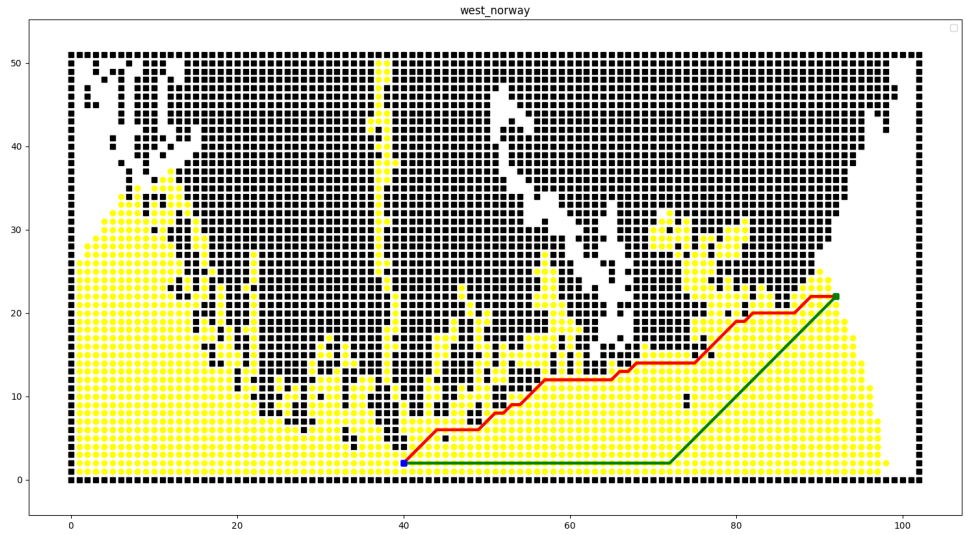


Figure 14: The red path represent the optimal route for 12,17,22,27,32 knot,The green path represent shortest distance route

From the result we could find that the optimal route for different velocity is the same as each other. And the ship velocity range is [12,17]knot. So the velocity will not influence the optimal path.Then I took a look in the fuel consumption. The result is shown as following

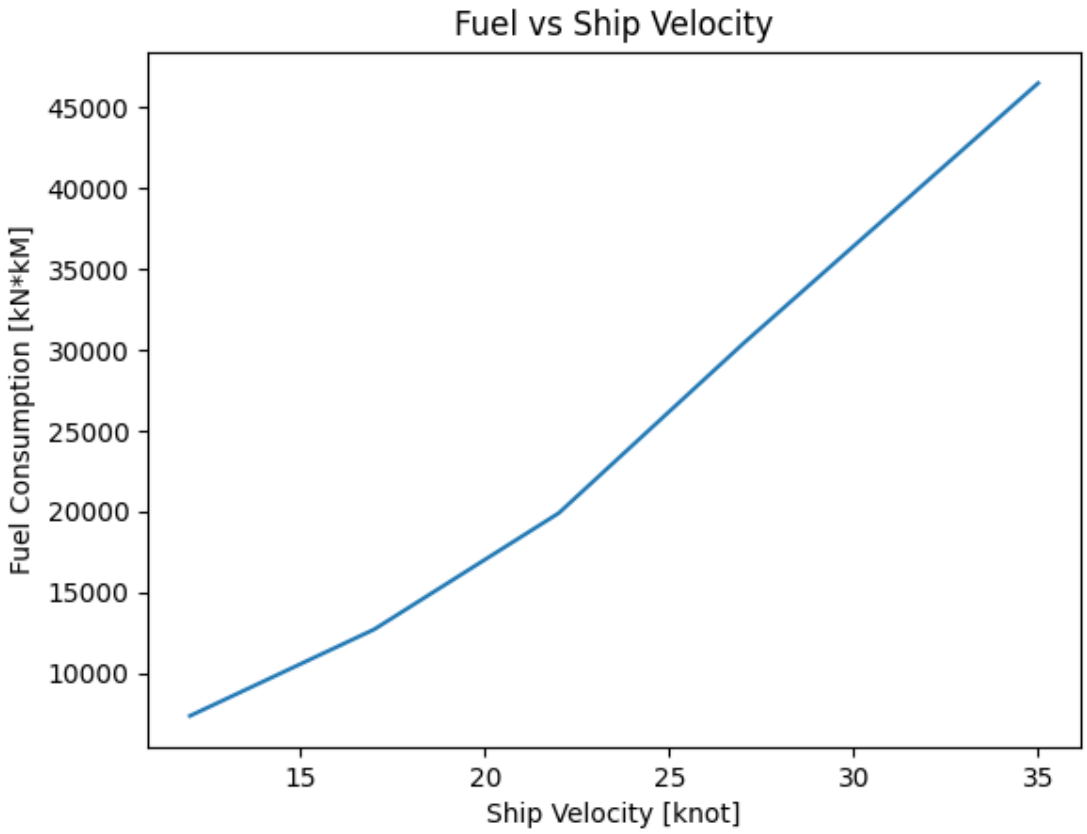


Figure 15: Fuel vs Ship Velocity

From the result we could find that the Fuel Consumption increase with the ship velocity.

5.3 Case Study C

In this case I would like to try to find put the influence of step length on the calculation time. So I selected the map of West Norway, 12 knot as ship velocity. The ship will travel from point (2,40) to point (82,16)
For the laptop device will influence much on the calculation time, So this set of experiment were all tested on my laptop, the Computer Configuration is attached as following table

Laptop	Lenovo Legion Y7000 (2019)
CPU	Intel Core i7-9750H
GPU	NVIDIA GeForce GTX 1660 Ti (Laptop)
RAM	16 GB
OS	Ubuntu 18

Table 5: Computer Configuration

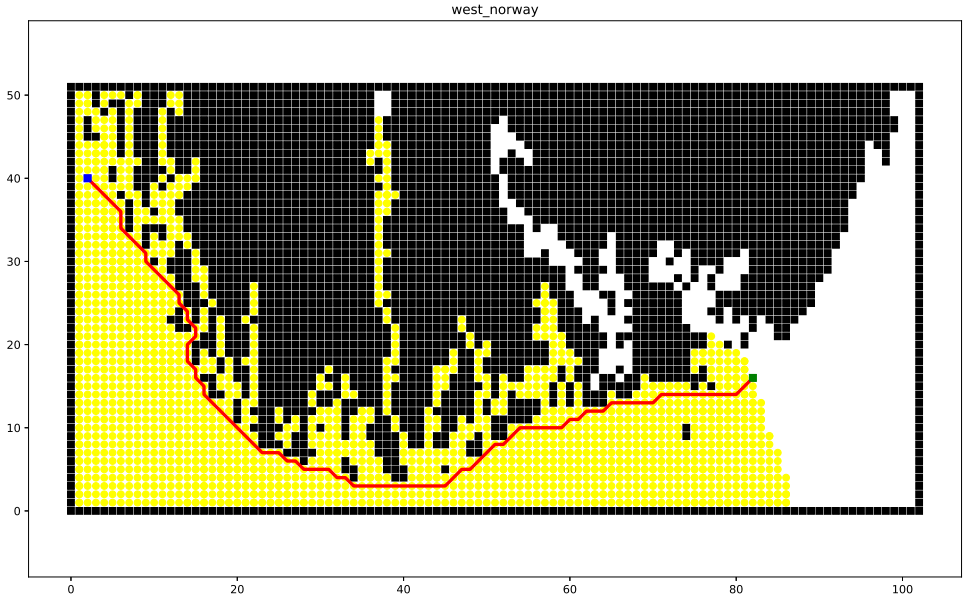


Figure 16: Step Length=1

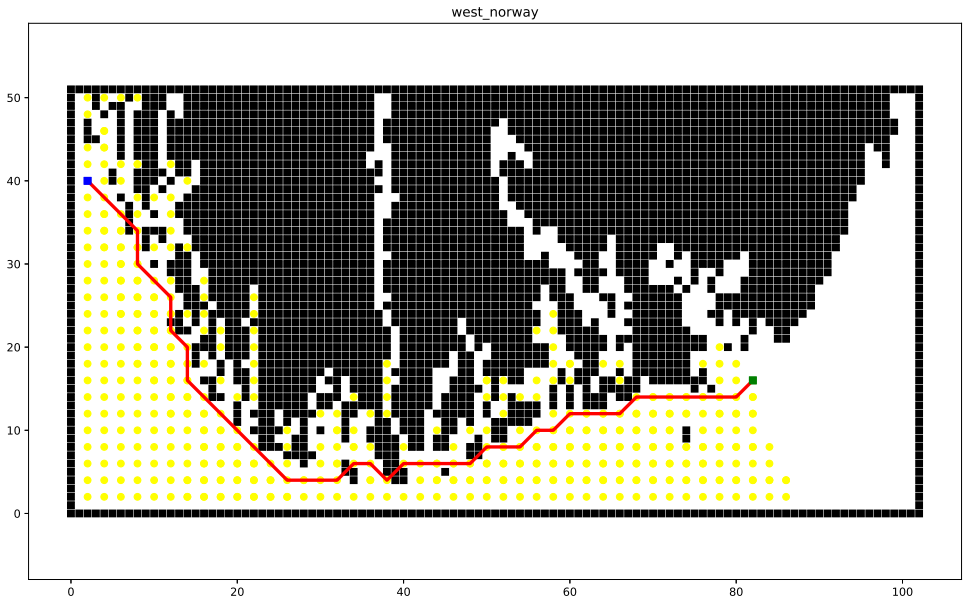


Figure 17: Step Length=2

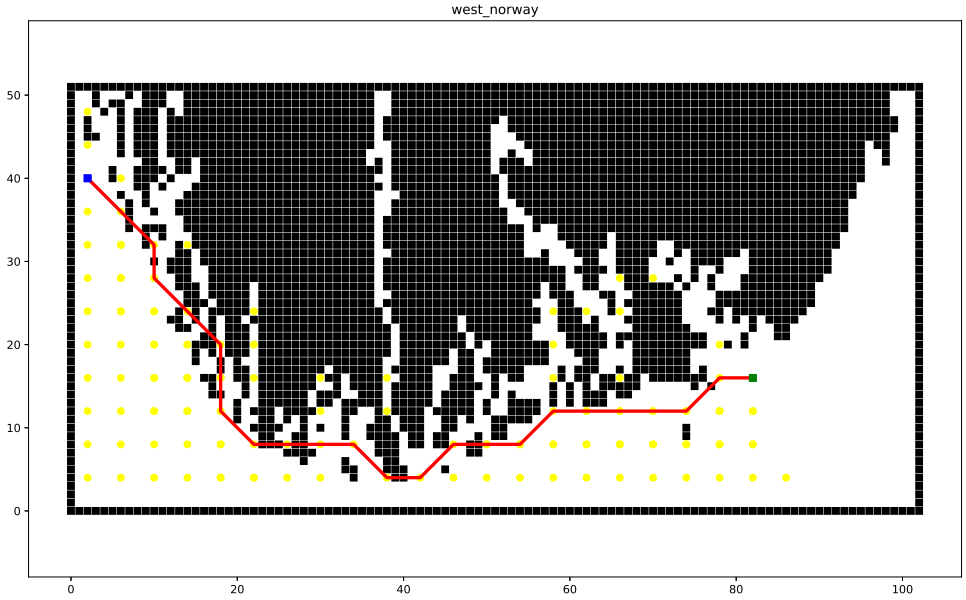


Figure 18: Step Length=4

Step Length	Calculation Time
L=1	240.18 [s]
L=2	53.27 [s]
L=4	7.52 [s]

Table 6: Step Length vs Calculation time

From the results we could find that the calculation time decrease exponentially with bigger step length in my laptop.Also the quality of the path is getting worse and worse. When step length is chosen as two, the path will transit some small obstacles.

5.4 Further Work

This program is not perfect. There are still some aspects could be improve. First of all, the weather dataset could be improve to online update directly instead of use the downloaded files. This program only use the Dijkstra's method. Some other method could be applied with this environment,and compare the performance of different method. Also Some other ship could be introduced into the ship model dataset.And compare the performance of different ship and their fuel consumption.

6 Conclusion

A program that solves Fuel consumption based ship routing problem have been developed throughout this project. The map and weather information of different locations in Norway are used to find out a optimal route from one point to the other. For better understand and easier to change parameters, A GUI is implemented in this project. The user can choose handful variables to find out the optimal route for difference cases. The results are visualized with plots.

Three cases are simulated in this project. In the first case, The program is tested in different maps of Norway. All results show a trend to follow the shape of land. In the second case, I try to find the relationship between velocity and optimal route, The results shows velocity will not influence much on route, but have big influence on the fuel consumption. In the final case, The relationship between Step length and calculation time are shown here. The larger step we choose, The shorter calculation time we need and worse quality of route we got.

This project is also a good method to improve the skill of python programming and a chance to combine the marine technology knowledge with programming.

References

Everybodywiki (n.d.), 'Pysimplegui'.
URL: *https://en.everybodywiki.com/PySimpleGUI*

Harris, C., Millman, K., Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M., Brett, M., Haldane, A., Río, J., Wiebe, M., Peterson, P. & Oliphant, T. (2020), 'Array programming with numpy', *Nature* **585**, 357–362.

Legovic, D. & Dejhalla, R. (2007), 'A comparison of a ship hull resistance determined by different methods', *Eng. Rev* **27**, 13–24.

Thorsen, S. H. L. (2019), Simulation based estimation of power consumption, Master's thesis, NTNU.

Wikipedia (2021), 'Dijkstra's algorithm — Wikipedia, the free encyclopedia', `http://en.wikipedia.org/w/index.php?title=Dijkstra's%20algorithm&oldid=1021617840`. [Online; accessed 06-May-2021].

Appendices

A GUI.py

```
# here is the main file of this program
import PySimpleGUI as sg
import numpy as np
import plotting
from plotting import Plotting
import matplotlib
import matplotlib.pyplot as plt
import Dijkstra
import time
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
class Ship_GUI():
    "here is a GUI for the ship routing system which contain a input window of..."
    "... the start point, end point,map selection,ship velocity and step length,."
    " A output window for plotting the map and the route of the result"
    def __init__(self):
        "here a intial frame of the gui "
        print('starting node')
        layout = [[sg.Text('Please select a map'),
        sg.Combo(['oslofjord','sorlandet','skagerrak',
                  'west_norway','n-northsea','nordland','troms-finnmark'],
                  default_value='west_norway',size=(15,1),key='map'),
        sg.Button('Change Map'),
        sg.T('Velocity'),
        sg.Input(key='Vel',default_text=12,size=(3,1)),
        sg.T('knot')],
        [sg.Text('Start Point: ',size=(15,1)),
        sg.Text('x:',size=(1,1)),
        sg.Input(key='start_x',default_text=1,size=(4,1)),
        sg.Text('y:',size=(1,1)),
        sg.Input(key='start_y',default_text=1,size=(4,1)),
        # sg.T('',size=(6,1)),
        sg.Text('End Point:',size=(15,1)),
        sg.Text('x:',size=(1,1)),
        sg.Input(key='end_x',default_text=2,size=(4,1)),
        sg.Text('y:',size=(1,1)),
        sg.Input(key='end_y',default_text=2,size=(4,1)),
        sg.Text('Step Length',size=(15,1)),
        sg.Text('L:'),sg.Input(key='L',default_text=1,size=(3,1))],
        [sg.Button('Start Calculation'),
        sg.Text('calculate time'),sg.Output(size=(15,1),key='TT')],

        [sg.Canvas(key='controls_cv')],
```

```
    [sg.T('Figure:')],
    [sg.Column(
        layout=[
            [sg.Canvas(key='fig_cv',size=(1500,900)
                    # it's important that you set this size
                    #size=(400 * 2, 400)
                    )]
        ],
        background_color='#DAE0E6',
        pad=(0, 0)
    )],
    ]


    self.window = sg.Window('Ship Routing GUI', layout,finalize=True)
    while True:

        self.event, self.values = self.window.read()
        if self.event=='Change Map':
            self.show_map()
        if self.event=='Start Calculation':
            self.cal_route()
        if self.event==sg.WIN_CLOSED:
            break
        draw_figure_w_toolbar(self.window['fig_cv'].TKCanvas, self.fig,
                              self.window['controls_cv'].TKCanvas)

    self.window.close
def show_map(self):
    plt.clf()
    start=(int(self.values['start_x']),int(self.values['start_y']))
    end=(int(self.values['end_x']),int(self.values['end_y']))
    ppl=plotting.Plotting(start,end,self.values['map'])
    self.fig = plt.gcf()
    DPI = self.fig.get_dpi()
    self.fig.set_size_inches(1500/ float(DPI), 900 / float(DPI))

    ppl.plot_grid(self.values['map'])

def cal_route(self):
    plt.clf()
    t1=time.time()
    start=(int(self.values['start_x']),int(self.values['start_y']))
    end=(int(self.values['end_x']),int(self.values['end_y']))
    location=self.values['map']
    L=int(self.values['L'])
```

```
        Vel=int(self.values['Vel'])
        dij = Dijkstra.Dijkstra(start, end,location,L,Vel)
        plot = plotting.Plotting(start, end,location)
        path, visited = dij.searching()
        t2=time.time()
        self.window['TT'].update(t2-t1)
        plot.plot_grid(location)
        fig = plt.gcf()
        DPI = fig.get_dpi()
        self.fig.set_size_inches(1500/ float(DPI), 900 / float(DPI))
        plot.plot_visited_in_GUI(visited)
        plot.plot_path(path)


def draw_figure_w_toolbar(canvas, fig, canvas_toolbar):
    if canvas.children:
        for child in canvas.winfo_children():
            child.destroy()
    if canvas_toolbar.children:
        for child in canvas_toolbar.winfo_children():
            child.destroy()
    figure_canvas_agg = FigureCanvasTkAgg(fig, master=canvas)
    figure_canvas_agg.draw()
    toolbar = Toolbar(figure_canvas_agg, canvas_toolbar)
    toolbar.update()
    figure_canvas_agg.get_tk_widget().pack(side='right', fill='both', expand=1)


class Toolbar(NavigationToolbar2Tk):
    def __init__(self, *args, **kwargs):
        super(Toolbar, self).__init__(*args, **kwargs)


if __name__ == '__main__':
    Ship_GUI()
```

B Dijkstra.py

```
import math
import heapq
import numpy as np
import plotting, Obstacles
import Resistance_Forces

import matplotlib.pyplot as plt
import get_grib
from get_grib import Get_wave_data,Get_wind_data
class Dijkstra():
    """
    Here is the Dijkstra program
    """
    def __init__(self, start, goal,location,L,Vel):
        self.s_start = start#start point
        self.s_goal = goal#  destination
        self.Obs = Obstacles.obstacles(location)
        self.Vel=Vel
        self.motion = [(-L, 0), (-L, L), (0, L), (L, L),
                       (L, 0), (L, -L), (0, -L), (-L, -L)]
        self.obs = self.Obs.obs  # position of obstacles
        self.wave_data=Get_wave_data(location).data
        self.wind_x,self.wind_y=Get_wind_data(location)
        self.range_x=np.size(self.wave_data,0)#import the data range from Obs
        self.range_y=np.size(self.wave_data,1)#
        self.OPEN = []  # priority queue / OPEN set
        self.Visited = []  # / VISITED order
        self.PARENT = dict()  # recorded parent
        self.g = dict()  # cost of the nodes
    def searching(self):
        """
        Here is the main loop of the Dijkstra program
        """

        self.PARENT[self.s_start] = self.s_start
        self.g[self.s_start] = 0
        self.g[self.s_goal] = math.inf
        heapq.heappush(self.OPEN,
                       (0, self.s_start))

        while self.OPEN:
            c, s = heapq.heappop(self.OPEN)
            self.Visited.append(s)
```

```
            if s == self.s_goal:
                break

            for s_n in self.get_neighbor(s):
                new_cost = self.g[s] + self.cost(s, s_n)

                if s_n not in self.g:
                    self.g[s_n] = math.inf

                if new_cost < self.g[s_n]:
                    self.g[s_n] = new_cost
                    self.PARENT[s_n] = s
                    heapq.heappush(self.OPEN, (new_cost, s_n))

        return self.extract_path(self.PARENT), self.Visited
    def get_neighbor(self, s):
        """
        find all the neighbor of current point
        """
        Neighbors=[]
        for i in self.motion:
            Neighbors.append((s[0] + i[0], s[1] + i[1]))
        return Neighbors

    def cost(self, s_start, s_goal):
        """
        Calculate Cost for this motion
        """
        if self.is_collision(s_start, s_goal):
            return math.inf
        if self.out_of_range(s_start,s_goal):
            return math.inf
        R_S=Resistance_Forces.Ship_Resistance()
        R1=R_S.hollenbach(self.Vel*0.5144)
        # R1=R_S.hollenbach_map(self.Vel)
        R2= R_S.Wave_resistance(self.wave_data[s_start[0]][s_start[1]])
        R3=R_S.Wind_resistance(s_start,s_goal,self.wind_x[s_start[0]][s_start[1]],
                               self.wind_y[s_start[0]][s_start[1]])
        R=R1+R2+R3
        W=R*math.hypot(s_goal[0] - s_start[0], s_goal[1] - s_start[1])
        return W
    def is_collision(self, start, end):
        """
        check if the start or end points are is collision with obstacles.
        """
```

```
        if start in self.obs or end in self.obs:
            return True
        return False
    def out_of_range(self,start,end):
        """
        check if the start or end points are getting out of the data range
        """
        if start[0]<0 or start[1]<0:
            return True
        if end[0]<0 or end[1]<0:
            return True
        if start[0]>self.range_x-1 or start[1]>self.range_y-1:
            return True
        if end[0]>self.range_x-1 or end[1]>self.range_y-1:
            return True
        return False

    def extract_path(self, PARENT):
        """
        from the end point getting back to the start point
        with parent set
        """
        path = [self.s_goal]
        s = self.s_goal
        while True:
            s = PARENT[s]#
            path.append(s)
            if s == self.s_start:
                break
        return list(path)

# for debug

if __name__ == '__main__':

    s_start = (1,1)
    s_goal = (35,5)

    dijkstra = Dijkstra(s_start, s_goal,"west_norway",1,12)
    plot = plotting.Plotting(s_start, s_goal,"west_norway")
    path, visited = dijkstra.searching()
    plot.plot_grid("West Norway")
    plot.plot_visited(visited)
    plot.plot_path(path)
    plt.show()
```

C get_grib.py

```
import pupygrib
import yaml
import os
def Get_wave_data(location):
    """
    Get the wave signficant height
    location: map name
    """
    nowdic=os.getcwd()#+'/'
    ship_file = open("Ship_Model.yaml")
    parsed_ship_file= yaml.load(ship_file,Loader=yaml.FullLoader)
    F=parsed_ship_file[location]
    Fname=os.path.join(nowdic,F[0])
    with open(Fname, 'rb') as stream:# signaficate wave hight
        for i, msg in enumerate(pupygrib.read(stream), 1):
            lons, lats = msg.get_coordinates()
            time = msg.get_time()
            values = msg.get_values()
            if i==63 : wave=values
    return wave

def Get_wind_data(location):
    """
    Get the wind velocity along x and y
    location: map name
    """
    nowdic=os.getcwd()
    ship_file = open("Ship_Model.yaml")
    parsed_ship_file= yaml.load(ship_file,Loader=yaml.FullLoader)
    F=parsed_ship_file[location]
    Fname=os.path.join(nowdic,F[1])
    with open(Fname, 'rb') as stream:# signaficate wave hight
        for i, msg in enumerate(pupygrib.read(stream), 1):
            lons, lats = msg.get_coordinates()
            time = msg.get_time()
            values = msg.get_values()
            if i==112: wind_u=values
            if i==113: wind_v=values
            #next row for debug
    return wind_u,wind_v
if __name__=='__main__':
    # F,M=Get_wind_data("nordland")
    # Get_wave_data("west_norway")
    Get_wind_data("oslofjord")
```

D Obstacle.py

```
import csv
import numpy as np
import get_grib
from get_grib import Get_wave_data
class obstacles:
    def __init__(self,location):
        self.Wave= Get_wave_data(location).mask
        self.x_range = np.size(self.Wave,0)  # size of background
        self.y_range = np.size(self.Wave,1)
        self.obs = self.obs_map()
    def obs_map(self):
        """
        Here is a map of land and data boundary
        """
        x = self.x_range
        y = self.y_range
        obs = set()

        for i in range(x):#left boundary
            obs.add((i, 0))
        for i in range(x):#right boundary
            obs.add((i, y - 1))

        for i in range(y):#up boundary
            obs.add((0, i))
        for i in range(y):#down boundary
            obs.add((x - 1, i))
        land=np.where(self.Wave==1.0 )# detect where is the land
        for m in range(np.size(land,1)):
            obs.add((land[0][m],land[1][m]))

        return obs
# for test use here
if __name__=='__main__':
    obstacles("west_norway").obs_map()
```

E plotting.py

```
import matplotlib.pyplot as plt
import Obstacles
import Dijkstra
from Dijkstra import Dijkstra

class Plotting:
    def __init__(self, start, end,location):
        self.start, self.end = start, end
        self.OBS = Obstacles.obstacles(location)
        self.obs = self.OBS.obs_map()

    def plot_grid(self, name):
        obs_x = [x[0] for x in self.obs]
        obs_y = [x[1] for x in self.obs]

        plt.plot(self.start[0], self.start[1], "bs")
        plt.plot(self.end[0], self.end[1], "gs")
        plt.plot(obs_x, obs_y, "sk")
        plt.title(name)
        plt.axis("equal")
    def plot_visited_in_GUI(self,visited):
        '''This function will plot the visited area directly '''
        if self.start in visited: visited.remove(self.start)
        if self.end in visited: visited.remove(self.end)
        visited_x=[visited[i][0] for i in range(len(visited))]
        visited_y=[visited[i][1] for i in range(len(visited))]
        plt.plot(visited_x,visited_y, color='yellow',marker= 'o',linestyle = 'None')
    def plot_visited(self, visited, cl='yellow'):
        if self.start in visited:
            visited.remove(self.start)
        if self.end in visited:
            visited.remove(self.end)
        count = 0
        for x in visited:
            count += 1
            plt.plot(x[0], x[1], color=cl, marker='o')
            plt.pause(0.001)
        plt.pause(0.01)
    def plot_path(self, path, cl='r'):
        path_x = [path[i][0] for i in range(len(path))]
        path_y = [path[i][1] for i in range(len(path))]
        plt.plot(path_x, path_y, linewidth='3', color=cl)
        plt.plot(self.start[0], self.start[1], "bs")
        plt.plot(self.end[0], self.end[1], "gs")
```

F Resistance_Forces.py

```
import yaml
import numpy as np
import get_grib
from get_grib import Get_wave_data,Get_wind_data
class Ship_Resistance():
    def __init__(self):
        """
         Here we will load the ship model scales from a ymal file
        when the ship not change these parameters will not change
        """
        ship_file = open("Ship_Model.yaml")
        parsed_ship_file= yaml.load(ship_file,Loader=yaml.FullLoader)
        self.Los= parsed_ship_file["Los"]
        self.TF= parsed_ship_file["TF"]
        self.TA= parsed_ship_file["TA"]
        self.Dp= parsed_ship_file["Dp"]
        self.Nrud= parsed_ship_file["Nrud"]
        self.NBrac= parsed_ship_file["NBrac"]
        self.NBoss= parsed_ship_file["NBoss"]
        self.NThr=parsed_ship_file["NThr"]
        self.L= parsed_ship_file["L"]
        self.Lwl = parsed_ship_file["Lwl"]
        self.B= parsed_ship_file["B"]
        self.CB = parsed_ship_file["CB"]
        self.S =parsed_ship_file["S"]
        self.A=parsed_ship_file["A"]
    def hollenbach(self,Vs):
        T= (self.TA+self.TF)/2
        Fi= (self.CB/self.L)*((self.B/2)*2*T)**(0.5)
        k= 0.6*Fi+145*Fi**3.5
        rho = 1025
        gravk = 9.81
        nu = 1.1395E-6
        self.Vs=Vs
        if self.Los/self.L<1:
            Lfn=self.Los
        elif self.Los/self.L>=1 and self.Los/self.L<1.1:
            Lfn=self.L+2/3*(self.Los-self.L)
        elif self.Los/self.L >= 1.1:
            Lfn= 1.0667*self.L
        else:print('shit')
        a =np.array([-0.3382 ,  0.8086  , -6.0258 ,
                     -3.5632, 9.4405 ,0.0146 ,0 ,0, 0, 0])
        b =np.array([-0.57424 ,         13.3893,        90.5960,
```

```
                4.6614,        -39.721,        -351.483,
                -1.14215        ,-12.3296,        459.254]).reshape(3,3)
    #b is 3 by 3 matrix
    d = np.array([0.854 ,-1.228 ,0.497])
    e = np.array([2.1701 ,-0.1602])
    f = np.array([0.17 ,0.20, 0.60])
    g = np.array([0.642, -0.635 ,0.150])
    Fn=Vs/(gravk*Lfn)**0.5
    dd=np.array([1, self.CB ,self.CB**2]).reshape(3,1)
    Fnkrit = np.array(d).dot(dd)
    c1 = Fn/Fnkrit
    # c1_min = Fn/Fnkrit

    Rns= Vs*self.L/nu
    CFs= 0.075/(np.log10(Rns)-2)**2
    CRFnkrit = max(1.0,(Fn/Fnkrit)**(c1))
    kL=e[0]*self.L**(e[1])#1 by 1
    Fnn= np.array([1 ,Fn, Fn**2]).reshape(3,1)
    #%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    # % Minimum values

    # % There is an error in the hollenbach paper and in Minsaas' 2003 textbook,
    # % is corrected in this formula by dividing by 10
    CRstandard = np.dot(dd.T,b).dot(Fnn)/10 # a scale
    big_array = np.array([T/self.B ,self.B/self.L ,self.Los/self.Lwl, self.Lwl/se
    big_array2 = (1+self.NThr)**a
    big_array_mean=np.prod(big_array)*np.prod(big_array2)
    CR_hollenbach= CRstandard[0]*CRFnkrit*kL*big_array_mean
    CR = CR_hollenbach*self.B*T/self.S
    #Resistance coefficient, scaled for wetted surface
    C_Ts = CFs +CR
    #Total resistance coeff. ship
    R_T_mean = C_Ts*rho/2*Vs*Vs*self.S#Total resistance to the ship
    return R_T_mean/1000
def hollenbach_map(self,Vs):

    """
    for speed up the calculation time I use a map
    (calculate the R_T_mean offline) instead of the
    last function
    """
    V_map=np.array(
    [12,12.1,12.2,12.3,12.4,12.5,12.6,12.7,12.8,12.9,13,13.1,13.2,13.3,
    13.4,13.5,13.6,13.7,13.8,13.9,14,14.1,14.2,14.3,14.4,14.5,14.6,14.7,14.8,
    14.9,15,15.1,15.2,15.3,15.4,15.5,15.6,15.7,15.8,15.9,16,16.1,16.2,16.3,
    16.4,16.5,16.6,16.7,16.8,16.9,17])
```

```
        F_map=np.array(
        [377.726,384.201,390.786,397.483,404.294,411.223,418.271,425.443,
                432.739,440.164,447.720,455.410,463.237,471.203,479.313,487.569,
                495.974,504.530,513.243,522.114,531.146,540.3446,549.710,559.248,
                568.962,578.854,588.928,599.188,609.637,620.279,631.118,642.157,
                653.399,664.850,676.512,688.390,700.487,712.807,725.354,738.132,
                751.146,764.399,777.895,791.639,805.635,819.887,834.400,849.177,
                864.223,879.543,895.140])
        ind=np.where(V_map==Vs)
        self.Vs=Vs
        return F_map[ind]


    def Wind_resistance(self,start,goal,V_wind_x,V_wind_y):
      """here we calculate the resistance of wind'''
        we only consider the force on surge direction of a ship
        """
        motion=np.array([goal[0]-start[0],goal[1]-start[1]])
        V_ship_vec=self.Vs*motion/np.linalg.norm(motion)
        V_wind_vec=np.array([V_wind_x,V_wind_y])*motion
        V_rel=V_ship_vec-V_wind_vec
        A=self.B*self.TA*0.8
        C_air=0.7
        R_wind= 0.5*1.29*A*C_air*(V_rel[0]*V_rel[0]+V_rel[1]*V_rel[1])
        return R_wind/1000

    def Wave_resistance(self,H_wave):
        """here we calculate the wave resistance"""
        R_wave = (1/16)*1025*9.8*H_wave*H_wave*self.B*np.sqrt(self.B/(self.L*0.15))
        return R_wave/1000
# for debug use
if __name__ == '__main__':
    H_wave=Get_wave_data("west_norway").data
    x, y= Get_wind_data("west_norway")
    V_wind_x=x.data
    V_wind_y=y.data

    shipR=Ship_Resistance()
    R_T=shipR.hollenbach(12*0.5144)# KN
    R_wind=shipR.Wind_resistance((5,5),(4,5),V_wind_x[5][5],V_wind_y[5][5])
    R_wave=shipR.Wave_resistance(H_wave[5][5])
    print(R_T)
    print(R_wind)
    print(R_wave)
```

G Ship_Model.yaml

```yaml
## ship model is set here


Los: 172.24   # Length over Surface [m]
TF: 10       # Draft at FP [m]
TA: 10        # Draft at AP [m]
Dp:  4       # Propelldiameter[m]
Nrud: 1      # Number of rudders
NBrac: 1     # Number of brackets
NBoss:  1    # Number of propeller boss
NThr:  1     # Number of tunnel thrusters
L: 162.45
Lwl: 172.24
B: 32
CB: 0.728
S: 8.267520000000000e+03
# here is the adress of the weather data
n-northsea: ['weather_dataset/n-northsea_wave.grb',
             'weather_dataset/n-northsea_wind.grb']
nordland: ['weather_dataset/nordland_wave.grb',
           'weather_dataset/nordland_wind.grb']
oslofjord: ['weather_dataset/oslofjord_wave.grb',
            'weather_dataset/oslofjord_wind.grb']
skagerrak: ['weather_dataset/skagerrak_wave.grb',
            'weather_dataset/skagerrak_wind.grb']
sorlandet: ['weather_dataset/sorlandet_wave.grb',
            'weather_dataset/sorlandet_wind.grb']
troms-finnmark: ['weather_dataset/troms-finnmark_wave.grb',
              'weather_dataset/troms-finnmark_wind.grb']
west_norway: ['weather_dataset/west_norway_wave.grb',
              'weather_dataset/west_norway_wind.grb']
```