# Simulation of Ice Management Operations


## Quentin Hisette


**Master Thesis**

presented in partial fulfillment
of the requirements for the double degree:
“Advanced Master in Naval Architecture” conferred by University of Liege
"Master of Sciences in Applied Mechanics, specialization in Hydrodynamics, Energetics and Propulsion” conferred by Ecole Centrale de Nantes

developed at University of Rostock
in the framework of the

**“EMSHIP”**
**Erasmus Mundus Master Course**
**in “Integrated Advanced Ship Design”**


Ref. 159652-1-2009-1-BE-ERA MUNDUS-EMMC


Supervisor: Prof. Robert Bronsart, University of Rostock

Reviewer: Prof. Mihaela Amoraritei, “Dunarea de Jos” University of Galati

Rostock, January 2014

## ABSTRACT

In the context of increasing industrial and economical activities in the Arctic Ocean, the present work focuses on physical ice management, the set of techniques used to reduce the threats from potentially hazardous or restrictive ice conditions relatively to a platform or structure.

The objective is to develop a MatLab simulation tool for ice management operations. According to several input parameters, the simulation code is able to predict an operation scenario, generate a visualization of the operation and produce post-processed results. Software structure is developed in the following.

First, the relevant parameters of the operating conditions are defined: ice thickness, drift velocity, icebreaking vessel power, etc. and maximum acceptable size of the broken floes. In level ice, the resistance is estimated thanks to the semi-empirical theory of Lindqvist (1989). The simulation tool determines then the equilibrium point of the icebreaker, i.e. the ship velocity such that propeller thrust equals ice resistance. For the resistance in ridges, a dynamic simulation model is developed from the theories of Slettebø & Ueland (2010) and Ehle (2011). Another part of the tool defines the turning ability of the icebreaking vessel(s). The code is based on a theoretical approach of the turning maneuver, coupled with the Lindqvist (1989) resistance estimation method in linear motion.

In order to manage the ice updrift a structure to defend, a management strategy (linear, sector, circular, ...) is selected according to the actual ice configuration : drift velocity, varying direction, etc. The area within the icebreaking vessel(s) will work is then defined from the dimensions of the protected structure, but also from the drift speed and directional variability. The trajectory of each vessel is calculated and an animation of the ice management operations is obtained, including drift motion of the ice pieces. If the requested ice management appears to be impossible, the software gives the reasons and suggests solutions (requested icebreaking power, ...). From the resulting broken ice field, the software measures the corresponding floe size, mass and momentum distributions that can then be used to compare the different management techniques, regards to the ice conditions and available icebreaking fleet.

Validation is performed through comparisons with full scale or model scale results. Thanks to the large database of experimental results of the Hamburg Ship Model Basin, several parameters of the calculation can be adjusted with values from similar vessels/maneuvers in order to improve the accuracy of the simulation.

The report ends with some conclusions and suggests further work to be performed in order to enhance the model.

# CONTENTS

# List of Figures

# List of Tables

## DECLARATION OF AUTHORSHIP

*I declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.*

*Where I have consulted the published work of others, this is always clearly attributed.*

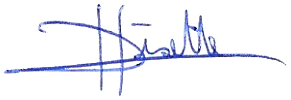
*Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.*

*I have acknowledged all main sources of help.*

*Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.*

*This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma.*

*Date: January 15, 2014* *Signature*



# INTRODUCTION

In Arctic regions, sea water can be covered, fully of partially, by many kinds of sea ice features, such as level ice, ice ridges and rubble fields. During winter and spring seasons, sea ice may cover very large areas of the Arctic Ocean. Additionally, ice pieces may drift and extent even more the coverage area where ice pieces can be found.

On the other hand, numerous industrial and economical activities take place in the Arctic Ocean, mainly shipping routes and hydrocarbon exploration and extraction. In order to protect against the drifting ice the offshore infrastructures requested for these activities to take place, ice forecasting, monitoring, hazard detection and clearing operations are generally needed (Fig. 1).

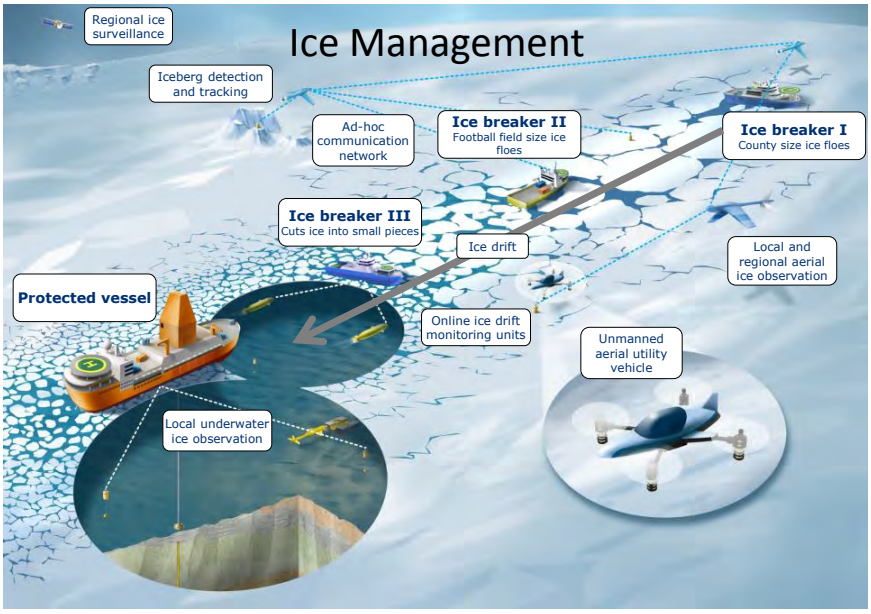



Figure 1: Ice management operations [1]

***Master's thesis***

In this context, the present Master's thesis focuses on the last aspect, ice clearing and breaking operations. Topic is the following :

*Simulation of Ice Management Operations*

The main objective is to develop a simulation tool for ice management operations, or more precisely, for *physical* ice management, i.e. the physical breaking of the drifting ice by means of icebreaking vessels. According to several input parameters, like ice distribution, vessel capabilities and requested target floe size, the simulation code will be able to predict an operation scenario, compute several working limits (maximum ice thickness, ...) and produce a visualization of the operations.

**The Hamburg Ship Model Basin** The resulting simulation tool has been developed for the Hamburg Ship Model Basin (HSVA) in order to meet the increasing demand from its clients. Thanks to the large database of experimental tests available in the company, the simulation parameters for any given management vessel(s) can be adjusted by comparison with experimental tests results for similar ships. This leads to more reliable results and more accurate simulations of ice management operations.

### *Implementation*

The simulation tool described in the present report has been written with the MatLab software. Its various built-in functions for file reading, data management, numerical resolution of non-linear systems, figure plotting and tools for animations were the main motivations for selecting such a software. The code itself makes use of a modular structure, where each module performs a specific calculation (ice resistance, maneuvering in ice, ...) and all the modules are interconnected by a simulation script.

The sources of the applied methods vary from one module to another. For ice resistance estimation and ridge breaking for example, the implemented methods are based on semi-empirical formulations from reference papers. In other cases, like the estimation of the vessel manoeuvrability in ice, a new method has been written in order to estimate the ice resistance on the vessel in turning motion, calculate the effective thrust and then determine the steady turning parameters. In both cases, the method has been systematically validated through comparison with experimental results.

### *Contents description*

A first part of the report is dedicated to the description of ice management operations. The current demand and evolution prospects are reviewed, and more details are given about the various ice management techniques that can be employed to efficiently defend an offshore structure against drifting ice features.

Then, the core part of the report is dedicated to the software description. The modular structure is defined and each of module is described. The general method is first exposed, then the main involved equations are given and the module is finally validated. Simulation scripts are then described for each ice management technique : vessel trajectory, area to be managed, requested vessel velocity, power level,... Finally, this part ends with the analysis of the resulting broken channel of the icebreaker : the distribution of the resulting floe size is calculated and compared to the initial target value of the calculation.

The report is concluded by a reminder of the main involved methods, a succinct description of the results and finally some conceivable future prospects of the present work.

# Part I
# ICE MANAGEMENT OPERATIONS

## 1 APPLICATION/DEMAND FOR ICE MANAGEMENT

During the last three decades, offshore hydrocarbon exploration beyond the Arctic circle has been increasing rapidly. Already in the 1970s, many wells were drilled in the shallow waters of the Beaufort Sea [2]. Mid-water depths ($30\,m$ – $70\,m$) waters have been explored a little bit later. Operating these wells was not requesting any specific protection from the drifting ice, as either the platforms were bottom-founded, or the wells were simply operated during the summer seasons only.

Nowadays, oil & gas producers are more and more looking for deep water drilling in the Arctic Ocean, beyond the continental shelf. In these areas, the ice-free waters season is often quite short and unpredictable. Additionally, the high water depths rule out the possibility of bottom-founded platforms, so that floating units must be used.

During operations (drilling, extraction,...), one of the keypoints is to manage a more or less stable position of the vessel, that is either moored to the sea bed, or fitted with a dynamic positioning system. In both cases, the use of ice management vessel(s), breaking the ice updrift the structure is necessary to reduce the loads on the structure. Without ice management, or when ice management is not sufficiently efficient, the floating platform has to disconnect from the well, inducing very important costs.

Simulation tools are one of the most interesting ways to improve the reliability of ice management operations. The objective is to minimize the frequency of unexpected disconnections due to failure of the ice management system while keeping the management fleet within reasonable costs. This demand considers the use of simulation tools at two different levels :

- The preliminary design phase of a platform. It is important to notice that floating platforms and ice management procedures should be designed together from start in order to have an accurate view of the future operational aspects of hydrocarbon extraction in the Arctic Ocean. The ice class of the platform is for example intimately related to the maximum load the hull can sustain, and then to the maximum floe size at given ice thickness, and finally therefore to the size of the requested ice management fleet.
- The operational planning of exploration and extraction. Depending on the ice and weather conditions, a simulation tool can be used to predict the requested number of ice management vessels and determine the most appropriated trajectory of the vessels.

The simulation tool developed in the present Master's thesis is aimed at these two tasks.

# 2 GENERAL DESCRIPTION OF ICE MANAGEMENT OPERATIONS

## 2.1 Definition

Ice Management is defined as the set of tasks consisting of (from Wright & Dunderdale [3]) :

- Ice monitoring and forecasting
- Ice hazard detection and tracking
- Ice alert system implementation
- Ice breaking and/or clearing

The present Master's thesis focuses on the last point, also called *physical ice management*. Physical ice management is used to reduce the threat from potentially hazardous or restrictive ice conditions relatively to a platform or structure.

A good example of ice management operation is the breaking of pack ice[1] that is drifting towards a floating platform or a FPSO vessel in order to reduce the loads on the structure and safely allow technical operations (oil extraction, ...) that would not be feasible otherwise, due to exceeding ice load levels. One or several icebreaking vessels are then used to destruct drifting level ice[2], ice floes[3] or ice ridges[4] into smaller size elements able to freely move around the structure to defend without inducing excessively large loads.

Fig. 2 shows several examples of ice management operations in the Arctic Ocean :

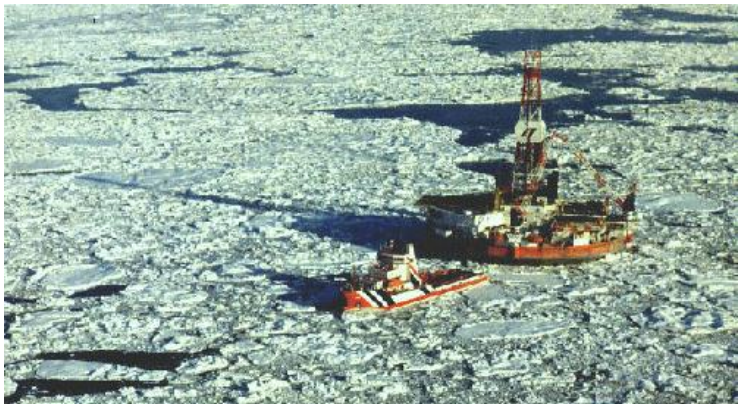
**Around a moored drilling vessel (Fig. 2a) :** In this type of operation, a drilling vessel is moored or uses dynamic positioning during drilling operations. Ice management vessels are breaking the ice in the updrift direction in order to decrease the loads on the drilling vessel. The objective is either to avoid exceeding loads on the mooring lines, or to reduce the required power for dynamic positioning.

**Around a floating drilling unit (Fig. 2b) :** Here, the drilling unit is a kind of barge (unpowered) designed for working in ice. Its hull (here circular) is reinforced to sustain higher ice loads and facilitates the flow of the ice pieces and mooring lines are protected from ice accumulation. It allows harder working conditions, but ice management operations around it are still required.

**Tanker escort (Fig. 2c) :** A third example consists in the ship escorting. Here, the pictures show a tanker being escorted by vessels up to an oil rig in the Arctic Ocean. When arrived close to the platform, the tanker is connected to the rig through a SALM buoy and the management vessels help it to maintain its position close to the buoy.

[1] Pack ice : Term used in a wide sense to include any area of sea ice, other than fast ice (i.e. sea ice that forms and remains fast along the coast), no matter what form it takes or how it is disposed [4].
[2] Level ice : Sea ice that is unaffected by deformation [4].
[3] Floe : Any relatively flat, isolated piece of sea ice [4].
[4] Ice ridge : Formation appearing when two ice sheets come into contact and break so that small pieces of ice accumulate along a line.

(a) Ice management around a moored drilling vessel [3]



(b) Ice management around a floating drilling unit [3]


(c) FSO tanker escorted up to, and then moored at a SALM buoy [3] [5]

Figure 2: Examples of ice management operations

## 2.2 Types of Ice Management Techniques

Depending on many parameters, like icebreaker capability, ice thickness or ice drift speed, the ideal ice management technique is different. In Fig. 3, the five basic pack ice management techniques that can be used in order to reduce the loads on a floating structure (offshore platform, FPSO, tanker, ...) are defined. These techniques are related to single vessel operations. When two or more management vessels are used, each of them can apply the same or a different technique.

**Linear technique :** When ice drift speed is high but drift direction rather constant, the ice management vessel uses a linear technique (Fig. 3a). The pack ice is broken in a straight lines pattern, parallely to the drift direction.

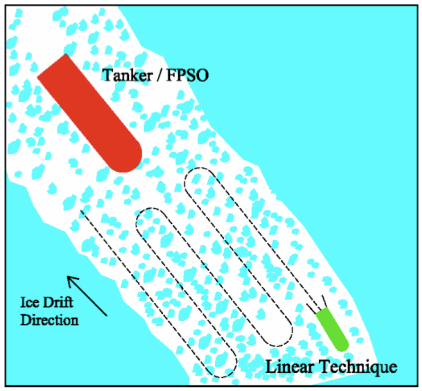
**Sector technique :** With the sector technique, the vessel breaks the ice perpendicularly to the drift direction, generating a wide managed ice area in front of the structure (Fig. 3b). This technique is to be preferred when ice drift speed is low and/or drift direction is unstable.

**Circular technique :** With this technique, the vessel moves in circular patterns updrift of the platform location (Fig. 3c). Diameter of the circles depends on the ice drift speed, and obviously on the vessel manoeuvrability. This technique is mainly used in case of high concentration of thin ice or small diameter but thick ice floes when drift direction is variable.

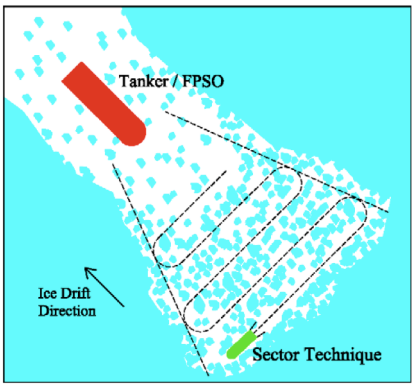
**Ice pushing :** Pushing ice is a rather different technique from the previous ones. It is used to remove large ice floes from the drift direction (Fig. 3d). The interest of this technique is that the threat to the platform is here fully removed from the drift line, on the contrary to a simple breaking of the floe, where the small floes may still cause problems. Pushing ice is sometimes performed by two or more vessels in order to prevent rotations of the floe.

**Propeller wash :** When the vessel is equipped with azimuth propellers, these can be directed at large angle outwards with full power and used to wash the ice on each side of the platform (Fig. 3e). The vessel is then stationary. This technique is very efficient to wash small pieces of thick ice, even in high concentrations.

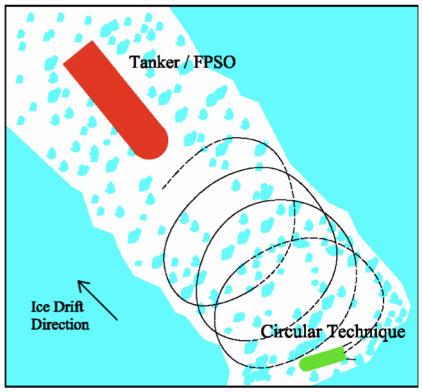
Please note that, however, the last two techniques are not simulated in the tool developed for the present Master's thesis, as they are too different from the three first ones. The linear, sector and circular techniques are more frequently used in the case of drifting level ice or drifting large floes, and simulation of these techniques is the main objective of the present thesis.



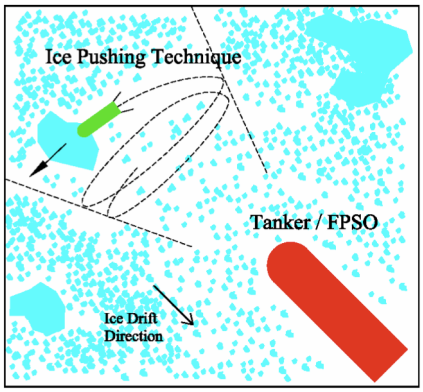

(a) Linear technique



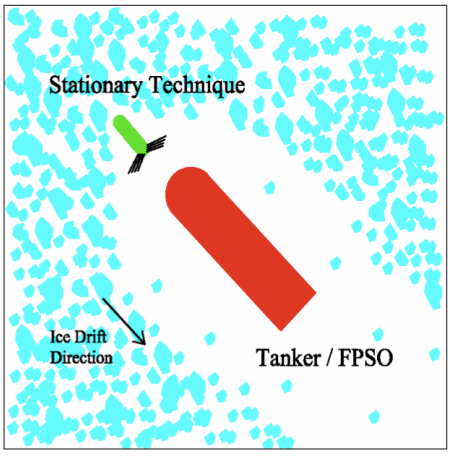

(b) Sector technique




(c) Circular technique




(d) Ice pushing technique




(e) Stationary technique (propeller wash)

Figure 3: Types of ice management techniques [3]

# Part II
# DEVELOPMENT OF ICE MANAGEMENT SIMULATION

## 1 STRUCTURE OF THE SOFTWARE

The MatLab code for simulation of ice management operations is written according to the structure proposed in the flow chart of Fig. 4. Each box consists in a different module, and the corresponding set of modules altogether form the simulation script.

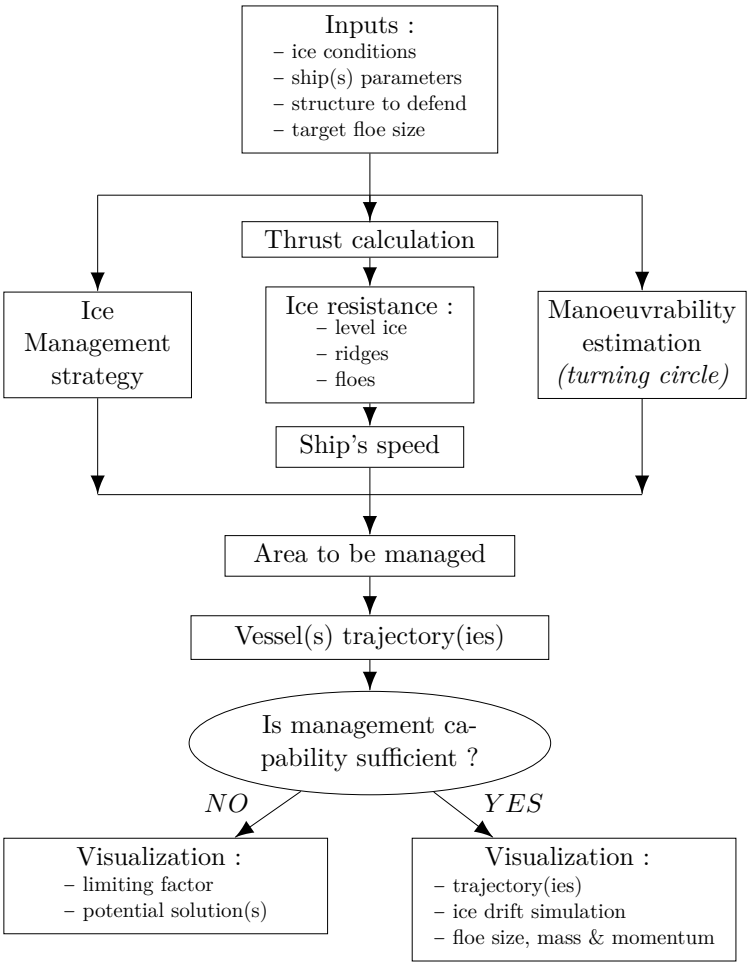



Figure 4: Flow chart of the simulation

**Inputs** From the study of the different ice management techniques, the relevant parameters to be considered in order to accurately define the operating conditions are defined. For example : ice thickness and drift velocity, managed ice concentration, icebreaking vessel power and dimensions, etc. The structure to defend is described by its main dimensions, and the maximum acceptable size of the broken floes that can hit the structure is considered as a target value of the calculation.

**Thrust calculation** According to the open-water characteristics of the icebreaking vessel(s) thruster(s), their thrust is evaluated as a function of the delivered power and a possible fraction of ice milling.

**Ice resistance** One of the main parts of the Master's thesis consists in the evaluation of the ice resistance. When moving in level ice or in large floes, the resistance is estimated thanks to the semi-empirical theory of Lindqvist [6]. For the resistance in ridges, the breaking process is more complicated and require a dynamic simulation model, as the vessel uses its kinetic energy to break the accumulation of ice. The method is based on some recent papers from Slettebø & Ueland [7] and Ehle-Myland [8] [9].

**Ship's speed** From a given delivered power and the resistance estimation, it becomes then possible to determine the equilibrium point of the icebreaker. In the simulation, the ship is power-driven, i.e. a power level is given as input and the software iterates until it gets the equilibrium velocity such that the propeller thrust equals the ice resistance.

**Manoeuvrability estimation** As the ice management operations mainly consist in zig-zag or closed trajectories, defining the turning ability of the icebreaking vessel is a crucial point. The objective of this section is to estimate how much time is required to accomplish a $90^o$ or a $180^o$ turn of any given radius. The code is based on a theoretical approach of the turning maneuver, coupled with the Lindqvist resistance estimation method in linear motion.

**Ice management strategy** In order to manage the ice updrift a structure to defend, several management strategies (or techniques) can be used (Fig. 3a to 3c). As each of them has its benefits and drawbacks, the best suited method has to be selected according to the actual ice configuration : drift velocity, varying direction, concentration, etc.

**Area to be managed** Once the management technique is defined, the area to be managed, that is to say, the area within the icebreaking vessel(s) will work, can be defined. The width and length of this area mainly depend on the protected structure dimensions, but also on the drift speed and on the variability of its direction.

**Vessel(s) trajectory(ies)** From the selected management technique, the trajectory of each vessel is then calculated in order to adequately cover the area to be managed.

**Visualization** The calculated trajectories are time-related, so that they can be used to visualize the vessel(s) motion in ice along time. From the ship's beam, the width of the broken channel is calculated and the drift motion of the ice pieces is simulated. As a result, an animation of the ice management operations is obtained.

**Floe size analysis** From the resulting broken ice configuration, the software is able to measure the corresponding floe size, mass and momentum and give the results under the form of an histogram. These results are then be compared to the input target floe size.

**Is management capability sufficient ?** At any moment of the process, the software should be able to inform the user if the requested ice management appears to be impossible. In this case, the software gives the reason of the problem (insufficient icebreaking power, excessive drift speed, ...) and suggests a few solutions (requested minimal icebreaking power, manageable floe size, ...).

**Results** Each the modules enumerated above will be described in the next sections of this report. It is important to notice that such a modular structure allows the software to be easily improved and modified by simply replacing any module by another or adding new modules in order to meet any specific demand from the Hamburg Ship Model Basin's clients.

# 2 APPLIED METHODS

## 2.1 Thrust Calculation

For the simulation of ship's motion, two main force components are always needed : thrust and resistance. This section is dedicated to the first component : from the propeller open-water characteristics, a given delivered power and the ship's velocity, the code should be able to determine the corresponding thrust and revolution rate :

$$[T_{net},\ N_{calc}] = Thrust\_calc\,(P_D,\ V,\ Open-water\ characteristics)$$

### *2.1.1 Propeller Open-water Characteristics*

The ship's propulsion system is mainly described by the propeller open-water characteristics. An example of such a data set is given in Fig. 5, with the advance number $J$, the thrust coefficient $K_T$ and the torque coefficient $K_Q$ defined as follows :

$$J = \frac{V_A}{N \cdot D} \qquad K_T = \frac{T}{\rho \cdot N^2 \cdot D^4} \qquad K_Q = \frac{Q}{\rho \cdot N^2 \cdot D^5}$$

with

- $V_A$, the advance velocity
- $N$, the revolution rate
- $D$, the propeller diameter
- $T$, the propeller thrust
- $Q$, the propeller torque

And the open-water efficiency $\eta_0$ is defined by

$$\eta_0 = \frac{P_T}{P_D} = \frac{T \cdot V_A}{Q \cdot 2\pi \cdot N} = \frac{K_T}{K_Q}\frac{J}{2\pi}$$

with $P_T$ the thrust power and $P_D$ the delivered power.

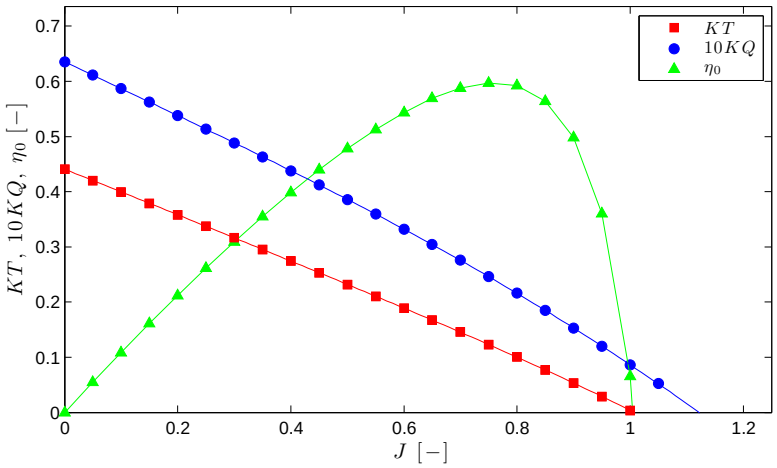



Figure 5: Open-water characteristics

However, the open-water characteristics are generally given under a tabular form, so that only discrete values of $K_T$, $K_Q$ are available. For this reason, a polynomial approximation is carried out in the MatLab script thanks to the `polyfit` function. Both $K_T$ and $K_Q$ curves can be confidently modeled by a $4^{th}$ order polynomial curve.

### *2.1.2 Wake Coefficient and Thrust Deduction Factor*

Prior to the net thrust calculation, two important coefficients must be evaluated.

**Wake coefficient** The wake coefficient $w$ expresses the influence of the hull shape on the arriving water velocity at the propeller :

$$V_A = (1 - w) \cdot V$$

Practically, the wake coefficient is generally given by an empirical formula depending on the ship's velocity. The script written for this master's thesis allows to possible formulae[5] :

$$w = w_1 + w_2^{V^{w_3}} \qquad (method\ 1)$$

$$w = \frac{w_1}{\tanh(V \cdot w_2)} \qquad (method\ 2)$$

where $w_1$, $w_2$ and possibly $w_3$ are empirical parameters. The typical curves related to these equations are given in Fig. 6a.

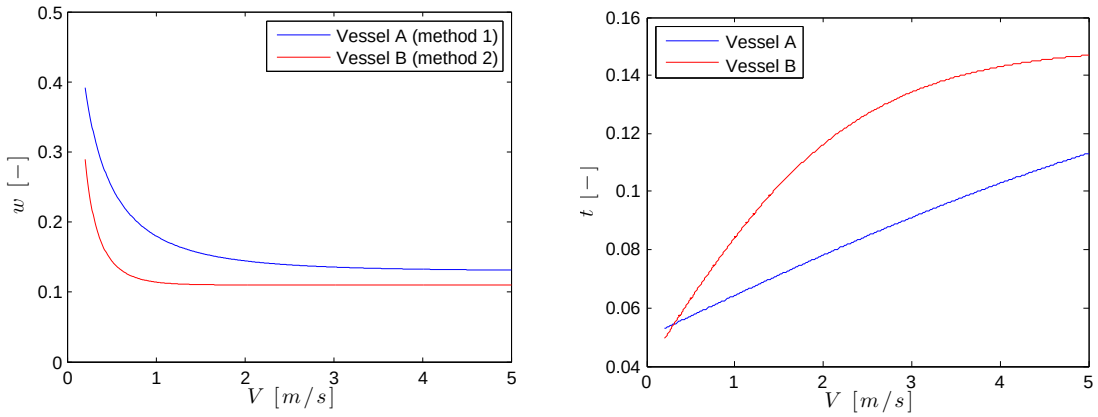
**Thrust deduction factor** Due to the propeller rotation, the water flow is accelerated, which causes an augmented resistance. In another point of view, the net thrust on the ship is less than the thrust generated by the propeller :

$$T_{net} = T_{prop} \cdot (1 - t)$$

with $t$ the thrust deduction factor. In the MatLab script, the following empirical formula[6] is considered for estimation of this factor :

$$t = t_1 + t_2 \tanh(t_3 \cdot \frac{V}{\sqrt{g \cdot L_{pp}}})$$

where $L_{pp}$ is the length between perpendiculars and $t_1$, $t_2$, $t_3$ are empirical parameters. Two typical curves of this thrust deduction factor are given in Fig. 6b.




(a) Wake coefficient (b) Thrust deduction factor

Figure 6: Wake coefficient and thrust deduction factor : typical curves

[5]Frequently used by the Hamburg Ship Model Basin.
[6]Idem [5].

### 2.1.3 Revolution Rate

The main difficulty of the thrust calculation is to find the equilibrium point, that is to say, the revolution rate $N$ that is both related to

- the appropriated delivered power (input) through $K_Q = \frac{Q}{\rho \cdot N^2 \cdot D^5}$ and $Q = \frac{P_D/Nbr}{2\pi N}$, where $Nbr$ is the number of propellers on the ship,

  AND

- a point of the fourth order polynomial approximation of $K_Q = aJ^4 + bJ^3 + cJ^2 + dJ + e$.

The solution of the problem satisfies :

$$\frac{P_D/Nbr}{2\pi \cdot \rho \cdot N^3 \cdot D^5} - a\left(\frac{V_A}{N \cdot D}\right)^4 + b\left(\frac{V_A}{N \cdot D}\right)^3 + c\left(\frac{V_A}{N \cdot D}\right)^2 + d\left(\frac{V_A}{N \cdot D}\right) + e = 0$$

This equation is numerically solved for $N = N_{calc}$, using the `fzero` MatLab function.

### 2.1.4 Total Net Thrust

From the equilibrium point $N_{calc}$, the advance number $J_{calc}$ is recalculated and the thrust coefficient is obtained from the $4^{th}$ order approximation previously determined :

$$K_T^{calc} = fJ_{calc}^4 + gJ_{calc}^3 + hJ_{calc}^2 + iJ_{calc} + j$$

The total thrust of all the propellers is then

$$T_{calc} = Nbr \cdot K_T^{calc} \cdot \rho \cdot N_{calc}^2 \cdot D^4$$

and the total net thrust is computed with the thrust deduction factor :

$$T_{net}^0 = T_{calc} \cdot (1 - t)$$

### 2.1.5 Propeller Ice Milling

In some configurations, the ice broken by the ship's bow can be sucked and milled by the propeller. The total net thrust on the ship is logically reduced due to the energy consumption of this milling process. This effect is taken into account by the following empirical formula, developed by the Hamburg Ship Model Basin :

$$T_{net} = T_{net}^0 \cdot \left(1 - 0.4 \cdot \frac{H_{milling}}{D} \tanh V\right)$$

where $H_{milling}$ is the estimated milled ice thickness. This value of the milled ice thickness is however difficult to estimate, as it strongly depends on the hull shape and therefore generally requests experimental tests to be evaluated.

## 2.2 Level Ice Resistance

This section is dedicated to the estimation of ship resistance in level ice. From the ship hull geometry, the ice properties and a given ship's velocity, the code should be able to determine the corresponding ice resistance :

$$R_{ice} = Ice\_resistance\,(V,\ Ship\ geometry,\ Ice\ properties)$$

### *2.2.1 Resistance Model*

The estimation of ship resistance in ice is a very particular subject, compared to the resistance in open water. The mathematical models developed for ship resistance in water cannot be used in ice conditions, as the process is completely different :

- a major part of the resistance comes from the ice breaking and crushing;
- the viscous drag cannot be estimated with classical formulations (ITTC formula, etc.), as the submerged part of the ship's hull is now in contact with ice pieces;
- the order of magnitude of the ship's velocity is different (relatively low compared to velocity in open water).

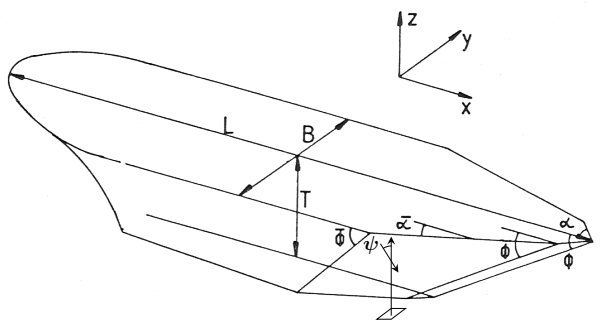
Therefore, particular mathematical models have to be developed in order to evaluate the resistance in ice. In this Master's thesis, the level ice resistance module is based on an improved version of the resistance model of Lindqvist [6], modified by HSVA. In this model, ship resistance is divided into the following components :

- Crushing resistance
- Bending resistance
- Submersion resistance

These terms will be evaluated in a quasi-static way, without any consideration of the ship's velocity. Then, they will be integrated in a speed dependent empirical formula.

### *2.2.2 Hull Geometry*

The model is developed for a wedge-shaped icebreaker, assuming a simplified model for the hull (Fig. 7).




$\phi$ : stem angle
$\alpha$ : waterline angle
$\psi$ : breaking angle, i.e. angle of vertical vector with normal to the surface
$\psi = \arctan \frac{\tan \phi}{\sin \alpha}$
$\bar{\phi}, \bar{\alpha}, \bar{\psi}$ : mean angles

Figure 7: Approximation of hull form (adapted from [6])

**1. Hull angles** Initially, only the mean values of the $\phi$, $\alpha$ and $\psi$ angles were used by Lindqvist [6], but the method has been improved by HSVA to represent more accurately the hull shape. As described in Fig. 8, the hull angles $\phi$ and $\alpha$ are measured in five positions : on the centerline, and at $\frac{1}{4}$, $\frac{2}{4}$, $\frac{3}{4}$ and $\frac{15}{16}$ of the ship's breadth[7]. The breaking angles $\psi$ are then calculated with

$$\psi = \arctan \frac{\tan \phi}{\sin \alpha}$$

so that the mean angles on each section of the beam can be obtained :

$$\phi^{av}_{1/4} = \frac{\phi_0 + \phi_{1/4}}{2} \qquad \alpha^{av}_{1/4} = \frac{\alpha_0 + \alpha_{1/4}}{2} \qquad \dots$$

**2. Wedge geometry** In order to improve the ice clearance under the hull, some icebreakers are equipped with a wedge, i.e. an extension of the keel forwards to the bow. This generates however an additional resistance, as will be described later (section II.2.2.7) as a function of the wedge length and the maximum wedge angle (Fig. 8).

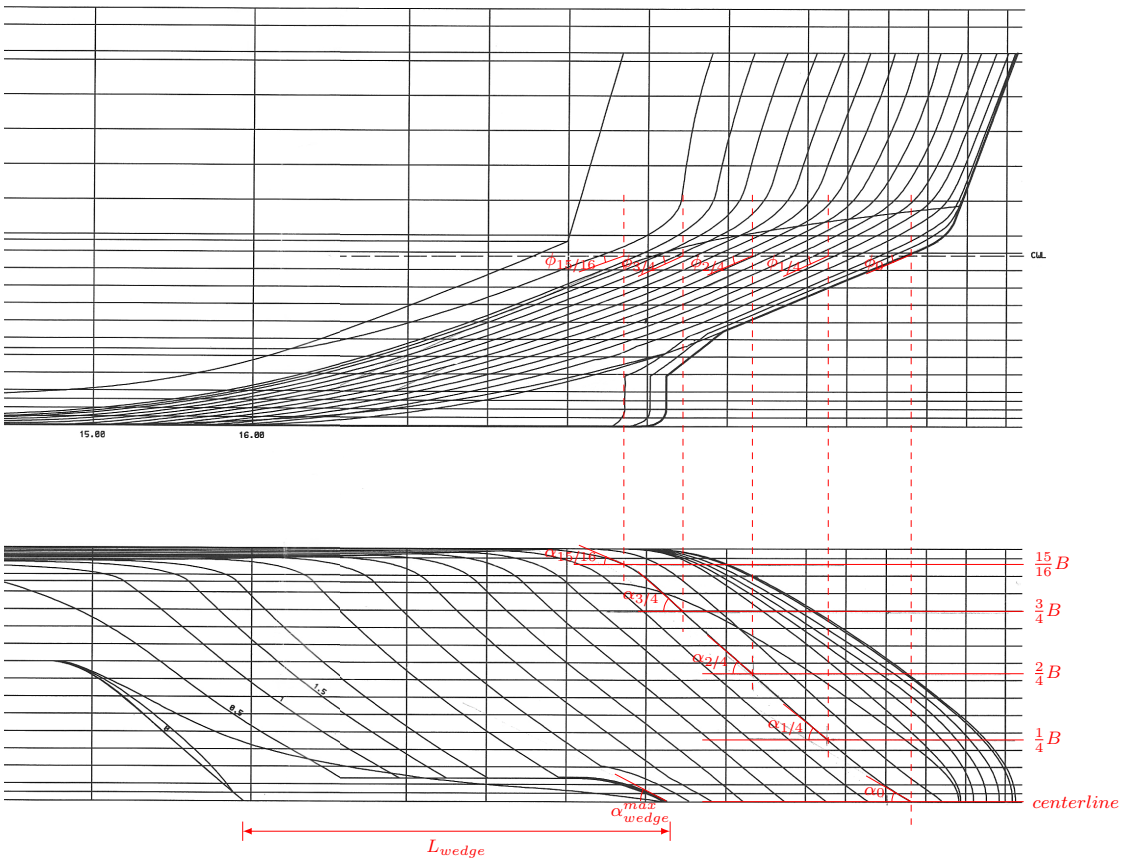
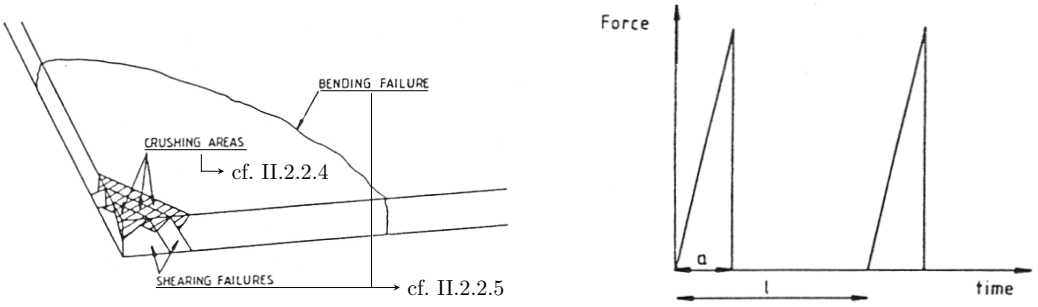



Figure 8: Hull angles

[7]The last measurement is taken in $\frac{15}{16}$ instead of $\frac{4}{4}$ to avoid $\alpha_{4/4} = 0^o$ and $\psi_{4/4} = 90^o$ which leads to instability further in the calculation.

### 2.2.3 Generalities

When the ship is moving forward to the ice sheet, the edge is crushed until the vertical force is large enough to induce shear cracking of small ice pieces. Then, at some distance aft from the stem, the stress becomes sufficiently large so that the ice sheet breaks by bending. The three phenomenons are described in the diagram of Fig. 9a. Their associated loads will be evaluated in the two following sections.




(a) Crushing, shearing, bending (adapted from [6])

(b) Time series of the vertical force during ice breaking [6]

Figure 9: Crushing, shearing and bending

### 2.2.4 Crushing at the Stem

In a wedge-shaped icebreaker, the force at the stem is never sufficiently large to break the ice by bending. In a first phase, the edge of the ice sheet is always crushed before to be broken by bending further aft of the ship. According to Lindqvist [6], the resistance component due to this crushing can be expressed by

$$R_c = F_v \frac{\tan\phi_0 + \mu\cos\phi_0/\cos\psi_0}{1 - \mu\sin\phi_0/\cos\psi_0} \qquad with \qquad F_v = \frac{1}{2}\sigma_b H_{ice}^2$$

where $F_v$ is the average vertical force on the ice, $\mu$ is the friction coefficient[8], $H_{ice}$ the ice thickness and $\sigma_b$ the ice bending strength.

### 2.2.5 Breaking by Bending

Diagram of Fig. 9b presents the evolution of the vertical force with time. When the crushing-shearing-bending process is ongoing, the force increases until breaking by bending. Then, force is zero[9] until the ship's stem reaches the new edge of the ice layer. This force should then be averaged to get the mean resistance force on the ship. This is what is carried out by Lindqvist in [6], and we get :

$$R_{b,i} = k \cdot \sigma_b \cdot \frac{B}{4}\frac{H_{ice}^3}{L_{cusp}^2} \cdot \left(\tan\psi_i^{av} + \frac{\mu\cos\phi_i^{av}}{\sin\alpha_i^{av}\cos\psi_i^{av}}\right) \cdot \left(1 + \frac{1}{\cos\psi_i^{av}}\right)$$

and the total bending resistance $R_b$ is then the sum over the four elements. In the previous expression,

- $k = \frac{3}{64} \approx 0.047$ is a calculation factor;

[8] The friction coefficient has generally a value of $\mu = 0.1$ for low friction paint and of $\mu = 0.16$ for normal antifouling paint.

[9] In practice, force is actually non zero, but relatively low compared to the maximum force at the bending level.

- $L_{cusp}$ is the length of ice cusp (see [6]), assumed to be one third of the characteristic length of ice :

$$L_{cusp} = \frac{L_{ch}}{3} = \frac{1}{3}\left(\frac{E \cdot H_{ice}^3}{12 \cdot (1-\nu^2) \cdot \rho_w \cdot g}\right)^{\frac{1}{4}}$$

- $B$ is the ship's breadth;
- $E$, $\nu$ and $\sigma_b$ are respectively the Young's modulus, Poisson's ratio and bending strength of ice, while $\rho_w$ is the water density.

### *2.2.6 Submersion Resistance*

When a ship is running in level ice, experimental tests and observations have shown that the submerged part of the hull was almost completely covered by ice pieces. As ice density is lower than water density, these ice pieces will generate a vertical lifting force on the hull. As a consequence, the normal force on the hull will induce a resistance force due to friction.

After some mathematical developments and estimation of the areas of the submerged hull, Lindqvist obtains the following expressions, where the submersion resistance is divided into a component $R_p$ due to the loss of potential energy and a component $R_f$ due to the frictional resistance :

$$R_p = \left(H_{ice} \cdot (\rho_w - \rho_i) + H_{snow}^{eff} \cdot (\rho_w - \rho_s)\right) \cdot g \cdot B \cdot T \frac{B+T}{B+2T}$$
$$R_f = \left(H_{ice} \cdot (\rho_w - \rho_i) + H_{snow}^{eff} \cdot (\rho_w - \rho_s)\right) \cdot g \cdot \mu \cdot (A_u + A_f \cos\bar{\phi} \cos\bar{\psi})$$

In the previous expressions,

- $\rho_w$, $\rho_i$ and $\rho_s$ are respectively the densities of water, ice and snow[10];
- $H_{snow}^{eff}$ is the effective thickness of snow to be considered in the calculation (generally, 50% of the total snow thickness is a good choice);
- $A_f$ and $A_u$ are rough approximations of respectively the bow and the ice covered bottom areas (with $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ the mean hull angles) :

$$A_f = B \cdot T \cdot \sqrt{\frac{1}{\sin^2\bar{\phi}} + \frac{1}{\tan^2\bar{\alpha}}} \qquad A_u = B \cdot (Cov_B \cdot L_{pp} - \frac{T}{\tan\bar{\phi}} - \frac{B}{4\tan\bar{\alpha}})$$

- $T$ and $L_{pp}$ are respectively the ship's draft and length between perpendiculars;
- $Cov_B$ is the proportion of bottom area covered by ice (generally between 50% and 70%).

### *2.2.7 Wedge Resistance*

As already explained in section II.2.2.2, some icebreakers are equipped with a wedge in order to improve the ice clearance under the hull. However, this element generates an additional resistance, estimated by HSVA with the following equation :

$$R_w = r_w \cdot V^2 \qquad with \qquad r_w = \frac{2}{3} \cdot B \cdot (H_{ice} \cdot \rho_i + H_{snow} \cdot \rho_s) \tan\alpha_{wedge}^{max}$$

[10]The snow layer above sea ice is here supposed to induce a submersion resistance on the ship, but was neglected for crushing and breaking resistance, as bending strength of snow is relatively low.

### *2.2.8 Open-water Resistance*

Even if it is relatively low compared to the ice resistance, the open-water resistance is not neglected in this calculation. In the developed MatLab script, the open-water resistance is supposed to be estimated from previous experimental tests or CFD analysis, producing a polynomial expression of the resistance versus velocity, valid for a minimum velocity $V_{ow}^{min}$. Between zero speed and $V_{ow}^{min}$, a $2^{nd}$ order interpolation is used to compute the resistance. The code has been developed to allow such a polynomial expression up to the sixth order, so that

$$R_{ow} = \begin{cases} aV^6 + bV^5 + cV^4 + dV^3 + eV^2 + fV + g & \text{if } V > V_{ow}^{min} \\ R_{ow}^{min} \cdot \frac{V^2}{V_{ow}^{min\,2}} & \text{if } V < V_{ow}^{min} \end{cases}$$

where $R_{ow}^{min}$ is the open-water resistance at $V = V_{ow}^{min}$ according to the polynomial expression. An example of such a curve is given in Fig. 10, where $V = V_{ow}^{min} = 3\,m/s$.

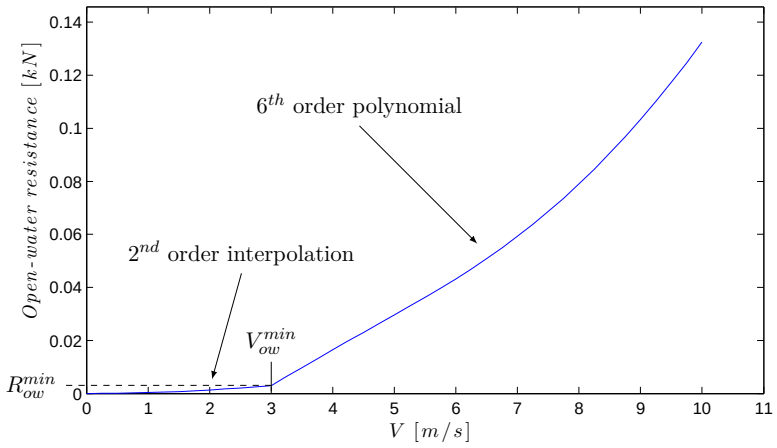



Figure 10: Example of open-water resistance curve

### *2.2.9 Speed Considerations*

The crushing, breaking and submersion resistance components have been given without any consideration of the ship's speed. Such a factor has to be inserted in the model. In ice conditions, resistance is rather large even for a quasi-static motion[11]. When speed increases, the following factors induce an increased resistance :

- increase of breaking and submersion resistance
- acceleration and ventilation of ice floes
- viscous drag

Except the last one, all these factors are specific to ice conditions. The wedge resistance and the open-water resistance are already function of the velocity. Lindqvist proposes the following empirical formula[12] to take consideration of the ship's speed $V$ :

$$R_{ice} = (R_c + R_b) \cdot \left(1 + 1.4\frac{V}{\sqrt{g \cdot H_{ice}}}\right) + R_s \cdot \left(1 + 9.4\frac{V}{\sqrt{g \cdot L}}\right) + r_w \cdot V^2 + \frac{2}{3} \cdot R_{ow}$$

in which we recognize the Froude numbers related to ice sheet and ship's length.

[11] At zero speed, the resistance may already represent 50% of its value at normal operating speed.

[12] It is generally considered that two thirds of the open-water resistance have to be added to the ice resistance in order to get the total resistance on the ship.

### *2.2.10 Equilibrium Point*

In the various simulation codes written for the present Master's thesis, the input of calculation is not the ship's velocity, but the delivered power. This is a more realistic approach, as the captain generally plays with the engine power to reach a given velocity.

For a given delivered power in level ice, an equilibrium has to be found for the ship's velocity. Mathematically, it corresponds to the velocity $V$ that makes the ice resistance to equal the total net thrust :

$$V \; ? \quad so\ that \quad T_{net} = R_{ice}$$

In MatLab, the equilibrium is reached thanks to the `fzero` function, looking for the zero of

$$f(V) = Thrust\_calc\,(P_D,\ V,\ Propulsion\,data) - Ice\_resistance\,(V,\ Ship\,geometry,\ Ice\,properties)$$

This calculation of the equilibrium velocity is implemented in the *Equil_level_ice* function given in Appendix D. As the numerical calculation is quite fast, this function can easily be used in any further iterative process in the various steps of the simulation (moving through ridges, manoeuvres, ...).

Finally, it is important to remember that the ice resistance at infinitely small speed is not zero. As a consequence, for too low delivered power, the equilibrium cannot be reached and the ship will be at rest, even if the thrust is non zero.

### *2.2.11 Validation*

The method for estimation of the ship resistance in ice can now be validated through comparison with experimental results. This section considers experimental data from three different ships that have been studied at the Hamburg Ship Model Basin. Their relevant dimensions are given in Table 1 (cf. Appendix A.1 for complete validation data) :

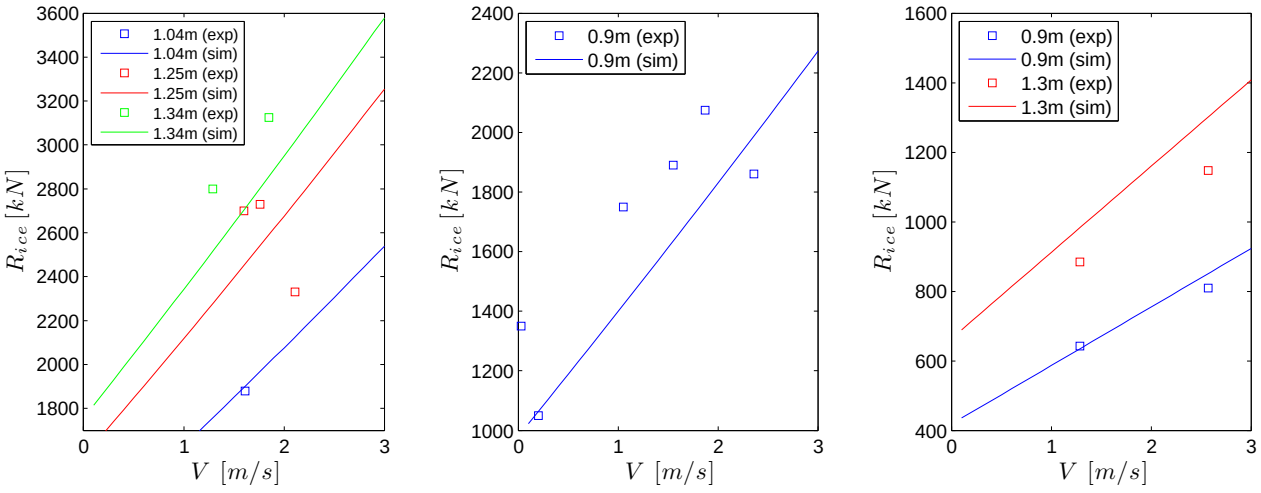
- Ship n$^o$1 and n$^o$2 are relatively large icebreakers, equipped with a wedge and good ice-breaking ability, especially ship n$^o$1 that has good (i.e. low) breaking angles $\psi$.
- Ship n$^o$3 is a smaller vessel, aimed to be used as an icebreaking emergency vessel.

These ships are comparable to those used for ice management operations, regarding their main dimensions and ice breaking capabilities. The testing conditions are also detailed in Table 1. Ships n$^o$1 and n$^o$2 were tested in ice with high bending strength, while ship n$^o$3 was tested in more common conditions, with a $20\,cm$ snow layer on top of moderately soft ice ($500\,kPa$).

With some additional assumptions for the estimation of open-water resistance and ice properties, the level ice resistance simulations have been performed. The results and comparison with model tests results are given in the three diagrams of Fig. 11. Globally, all the simulations exhibit results in good agreement with the experiments. The slope of the curve is very similar, while the numerical values seem to be slightly below the experiments for ships n$^o$1 and n$^o$2, and above for ship n$^o$3.

Table 1: Level ice resistance : validation data

| Vessel | | Ship n$^o$1 | Ship n$^o$2 | Ship n$^o$3 |
|---|---|---|---|---|
| $L_{pp}$ | $[m]$ | 184.3 | 126.6 | 72.9 |
| $B$ | $[m]$ | 28.9 | 23 | 18.4 |
| $T$ | $[m]$ | 9.5 | 7.5 | 6.52 |
| $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ | $[deg]$ | [23.6, 36.2, 37.1] | [29.0, 32.9, 52.54] | [26.2, 36.6, 40.3] |
| $m_{ship}$ | $[ton]$ | 38945 | 15555 | NA |
| $H_{ice}$ | $[m]$ | 1.04, 1.25, 1.34 | 0.90 | 0.90, 1.30 |
| $H_{snow}$ | $[m]$ | 0.0 | 0.0 | 0.20 |
| $\sigma_b$ | $[kPa]$ | 1450 | 1100 | 500 |




(a) Ship n$^o$1 (b) Ship n$^o$2 (c) Ship n$^o$3

Figure 11: Level ice resistance : validation

## 2.3 Resistance in Floes

When a ship advances in floes-covered water or tries to break further previously broken floes, the ice resistance can be significantly lower than the one that would be observed in level ice conditions. It is therefore important to estimate this resistance in floes in order to get an accurate value of the requested power level when managing ice in floes.

### *2.3.1 Method*

The objective of this part of the code is to evaluate the ice resistance of the ship as a function of the ice floes size $d_{floe}$ and ice concentration $C_{ice}$ :

$$R_{ice} = Ice\_resistance\_floes\,(V,\ d_{floe},\ C_{ice},\ Ship\ geometry,\ Ice\ properties)$$

From simple deductions, it results that the resistance in floes should be (Fig. 12) :

- Equal to the level ice resistance for an infinite floe size and fully covered surface
- Equal to the brash ice resistance for very small floes
- Rapidly decreasing for decreasing ice concentration[13]
- Equal to the open-water resistance for zero ice concentration

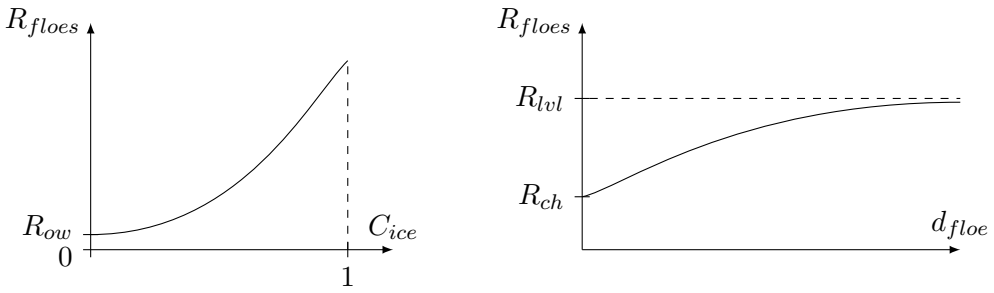



Figure 12: Estimation of the ice resistance in floes

### *2.3.2 Broken Channel Resistance*

The channel resistance is the lower limit of the ice resistance in floes. When the floes are becoming too small, they can be assimilated as brash ice (for 10/10 ice concentration). According to Riska et al. [10], the channel resistance in brash ice can be expressed by

$$R_{ch} = C_1 + C_2 + C_3 C_\mu (H_F + H_{ice})^2 (B + C_\psi H_F) + C_4 L_{par} H_F^5 + C_5 \left(\frac{LT}{B^2}\right)^3 \frac{A_{wf}}{L} \left(\frac{F_N}{F_N^0}\right)^2$$

where

- $C_1 = C_2 = 0$ for all icebreaker classes except IA Super[14], $C_3 = 845\,kg/m^2s^2$, $C_4 = 42\,kg/m^2s^2$ and $C_5 = 825\,kg/s^2$ are calculation constants
- $C_\mu$ and $C_\psi$ are empirically determined coefficients :

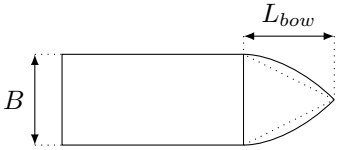
$$C_\mu = \max\left(0.15\cos\phi_{2/4} + \sin\psi_{2/4}\sin\alpha_{2/4},\ 0.45\right)$$

[13] When the concentration decreases, instead of breaking, the floes are more easily pushed on the sides of the vessel, which results in a lower resistance.

[14] Please refer to [10] for IA Super class.

$$C_\psi = \begin{cases} 0 & if\ \psi_{2/4} \leq 45^o \\ 0.047 \cdot \psi_{2/4} - 2.115 & otherwise \end{cases}$$

- $H_F = 0.26 + \sqrt{H_{ice} B}$, with $H_{ice}$ the ice thickness
- $L$, $B$ and $T$ are respectively the ship's length, beam and draft. Additionally, the ratio $LT/B^2$ should be taken so that
$$5 \leq \left(\frac{LT}{B^2}\right)^3 \leq 20$$
- $A_{wf}$ is the bow waterline area, roughly estimated as follows :




$$A_{wf} \approx 1.2 \cdot L_{bow} \cdot B$$

- $F_N = \frac{V}{\sqrt{gL_{pp}}}$ and $F_N^0 = \frac{2.57}{\sqrt{gL_{pp}}}$ are the ship's Froude numbers for respectively the actual and a reference speed of $2.57\,m/s$ $(5\,knots)$.

### *2.3.3 Resistance in Floes*

According to [11], and considering that the resistance in floes cannot be less than the open-water resistance $R_{ow}$, the curve of the resistance in floes can then be estimated as a function of the ice concentration $C_{ice}$ and the floe size $d_{floe}$ with the following formula :

$$R_{floes} = \max\left[\left(\frac{10\,L_{pp} - d_{floe}}{10 L_{pp} - 2} R_{ch} + \sqrt{\frac{d_{floe} - 2}{10\,L_{pp}}} R_{lvl}\right) \times C_{ice}^3\ ;\ R_{ow}\right]$$

where $R_{lvl}$ is the ice resistance in level ice for the same velocity. This formula must be used with $d_{floe}$ so that

$$2 \leq d_{floe} \leq 10\,L_{pp}$$

Outside the range, $d_{floe}$ is given the upper/lower endpoint value. As a result, for $C_{ice} = 10/10$ (full ice coverage), the floe resistance equals $R_{ch}$ for very small floes ($d_{floe} \leq 2\,m$) and almost equals $R_{lvl}$ for very large floes $d_{floe} > 10\,L_{pp}$. The proportionality with $C_{ice}$ to the third power comes from analysis of experimental results. An example of the resistance curve in floes for different ice concentrations is given in Fig. 13.

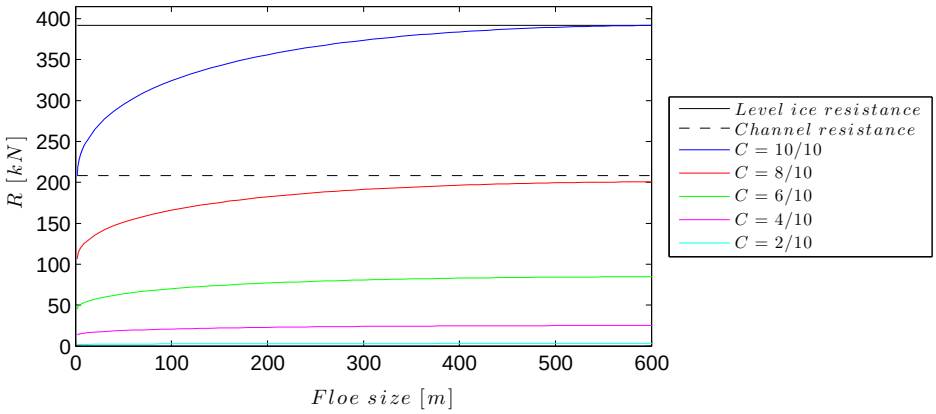
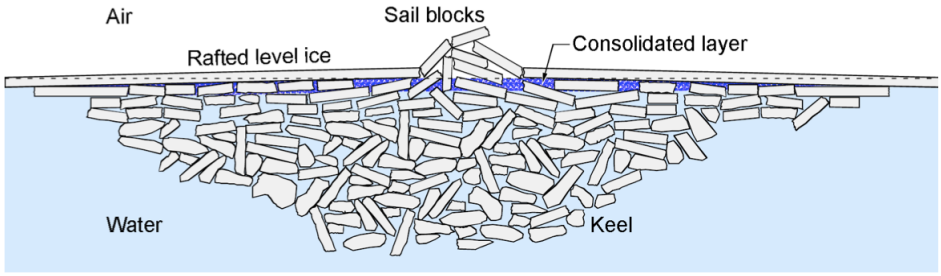



Figure 13: Ice resistance in floes : influence of floe size and ice concentration

## 2.4 Moving through Ice Ridges

Ice ridges are formations that consist in a line or a wall of broken ice forced by pressure. In other words, when two ice sheets come into contact, they break and small pieces of ice accumulate along a line. They form then a bi-triangular structure, with sail above water level and submerged ridge keel, as described in Fig. 14a. After formation, the ice blocks may refreeze together, forming a solidified structure drifting with the surrounding ice sheet. A typical ice ridge is shown in the picture of Fig. 14b.




(a) Structure [12]

(b) Typical sea ice ridge [7]

Figure 14: Ice ridges

Sea ice ridges are one of the most difficult obstacles encountered in the Arctic Ocean. Ice management vessels have to cut them into many smaller pieces of acceptable size for the structure to defend. A specific section of the present report is dedicated to the breaking of such ice formations.

From the ridge geometry and the initial delivered power of the ship, the objective is to compute the time series for the motion, the ice resistance and the thrust during the advance of the ship through the ridge :

$$[x(t),\ \dot{x}(t),\ \ddot{x}(t),\ R_{ice}(t),\ T_{net}(t)] =$$

$$Ice\_ridge\,(P_D^0,\ Ridge\ data,\ Ice\ properties,\ Propulsion\ data,\ Ship\ geometry)$$

The numerical simulation described below is based on the theoretical developments of Mellor (2008, [13]) and Ueland, Slettebø (2010, [7]), and on the experimental tests of Ehle-Myland (2011, [8] and 2013, [9]) performed at the Hamburg Ship Model Basin.

### *2.4.1 Schematic View of the Breaking Process*

Let us consider a ship going through an ice ridge surrounded by level ice. According to many experiments ([8], [9]), the ice resistance can be schematically described in six stages (Fig. 15) :

1. In order to gain kinetic energy, the ship accelerates at full power from its initial speed until she reaches her maximum velocity in level ice. The ice resistance thus increases and then keeps a constant value.

2. At beginning of stage 2, the ship's bow impacts the ridge. Ice resistance increases rapidly up to a maximum, when the fore shoulder attains the maximum ridge keel depth.

3. In the next stage, the bow is leaving the ridge, while the parallel midbody comes more and more in contact with the ridge. Altogether, this results in a decrease of the resistance up to the so-called midbody resistance.

4. In stage 4, the fore shoulder is already back in level ice, so that only the midbody is in contact with the ridge. No change is observed in the contact areas, so that resistance is constant during this phase.

5. During stage 5, the parallel midbody is leaving the ridge and the stern comes in contact. The midbody resistance decreases, while the stern resistance increases due to the appendages. Altogether, the ice resistance decreases smoothly up to level ice resistance.

6. Finally, the ship is back in level ice, but keeps full power in order to recover speed. Then, the power is decreased to its initial level and we go back to the initial conditions.

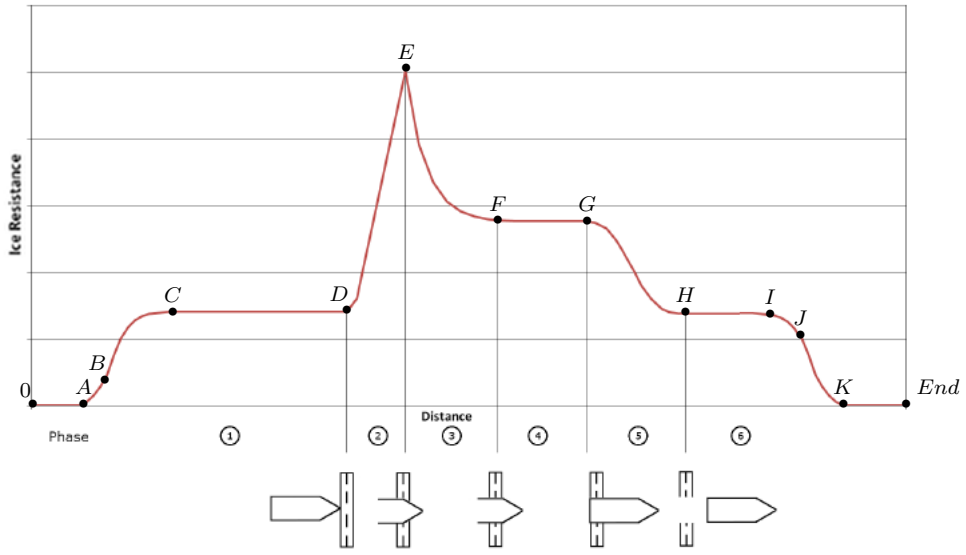


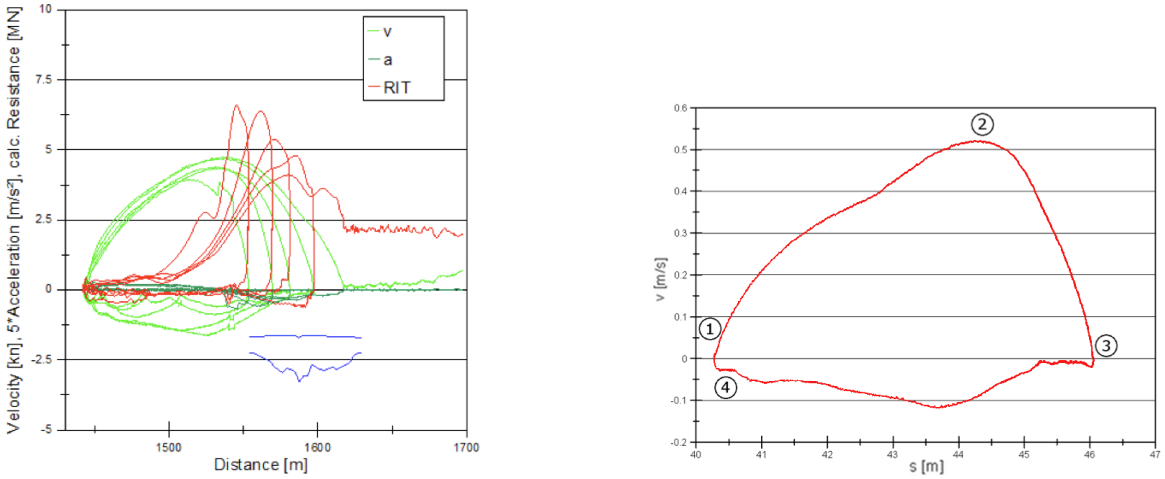

Figure 15: Schematic breaking of an ice ridge (modified from [9])

### *2.4.2 A Few Words about Ramming...*

When the kinetic energy of the ship is not sufficient to break the ridge at the first advance, the ship has to do a ramming process, that is to say, going through the ridge in several successive steps. An example of such a process is given in Fig. 16a, where 5 rams are required by the ship to pass through the ridge.

A typical ramming cycle is described in Fig. 16b. From ① to ②, the ship accelerates in level ice in order to increases its kinetic energy. She hits the ridge in ② and the energy is consumed to break the ice. If the combination of kinetic energy and propeller thrust is not sufficient, the velocity reaches 0 in ③, so that the ship has to move back to its initial position and start a new ramming cycle. On the other hand, if this combination of kinetic energy and propeller thrust is large enough, the ship can break the ridge at the first advance.

In the script described in the present report, only the second possibility is modeled. The ridge is considered to be relatively small compared to the ship, so that it can be broken in one run. The ramming process has not been modeled, mainly because it requires some additional research to evaluate the time required by the ship to move back after being stuck in the ridge, and to model a backwards motion in an ice broken channel (from ③ to ④, on the diagram of Fig. 16b). Modeling such a process could however be a very interesting extension to the present work.




(a) Example of ridge breaking in ramming mode [8]

(b) Typical pathway of ramming cycle [8]

Figure 16: Ridge breaking in ramming mode

### *2.4.3 Numerical Simulation*

This subsection presents step by step the modeling process that has been used to represent the six stages of the motion through a ridge. In order to facilitate the numerical implementation, these six stages have been split into twelve steps, as described on the result example of Fig. 17. Each of these steps is characterized by a specific ice resistance and/or power level. All of them have their own starting and ending conditions, the ending condition of one step being the starting condition of the next one.

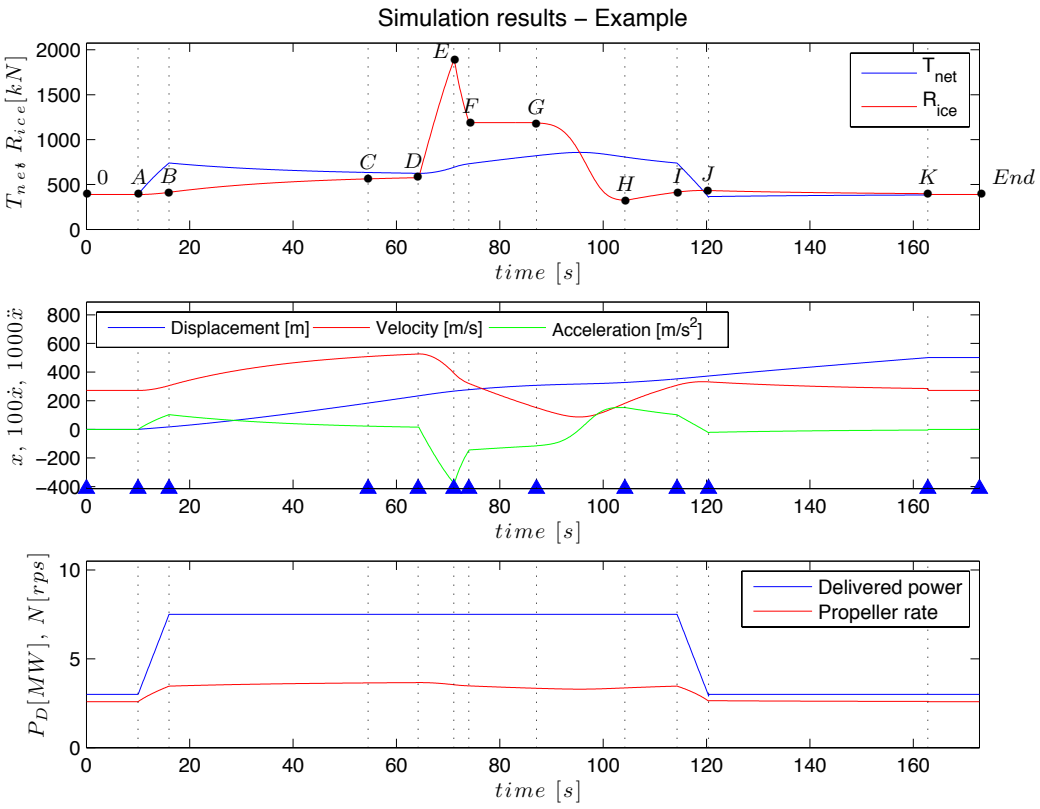



Figure 17: Example of a numerical implementation of an ice ridge breaking

***Acceleration, velocity and position***

The code written for this simulation considers a dynamic model of the ship. The actual thrust of the ship is generally not large enough to sustain the ice resistance at the top of the curve. For example, on the simulation diagrams of Fig. 17, the thrust in $E$ is only 1/3 of the ice resistance at the same instant. Actually, the step $C-D$ is used by the ship to accelerate until her maximum velocity in level ice, so that her kinetic energy is maximum prior to hit the ridge.

For this reason, a dynamic model is used to compute at each time step $k$ the new acceleration, velocity and position :

$$\ddot{x}^k = \frac{1}{(1+c_{m_{add}})\cdot m_{ship}} \cdot (T_{net}^k - R_{ice}^k)$$
$$\dot{x}^k = \dot{x}^{k-1} + \ddot{x}^k \cdot dt$$
$$x^k = x^{k-1} + \dot{x}^k \cdot dt$$

where

- $k$ indicates the current time step, and $k-1$ the previous one;
- $dt$ is the time step;
- $m_{ship}$ and $c_{m_{add}}$ are respectively the ship's displacement and the percentage of added mass to be considered for surge motion;
- $T_{net}$ is the net thrust acting on the ship, calculated from the actual conditions (delivered

power, ship's velocity,...) with the thrust calculation function (cf. section II.2.1):

$$[T_{net},\ N_{calc}] = Thrust\_calc\,(P_D,\ V,\ Propulsion\,data)$$

In this function, the input delivered power depends on the current stage, while $V$ is the velocity at the previous time step : $V = \dot{x}^{k-1}$;

- $R_{ice}$ is the ice resistance at the current position of the ship, as it will be described in the following subsections.

***Step by step modeling***

The method used to model each step of the process is then described, considering the delivered power, the thrust calculation, the ice resistance and the condition to jump from one step to another.

**0-A** This initial step is optional, as it consists only of a waiting phase before approaching the ridge. The ship is in level ice, starting with the initial conditions :

$$\begin{cases} \ddot{x} = \ddot{x}^0 = 0 \\ \dot{x} = \dot{x}^0 = V_0 \\ x = x^0 = 0 \end{cases} \quad and \quad \begin{cases} P_D = P_D^0 \\ T_{net} = T_{net}^0 \\ R_{ice} = R_{ice}^0 \end{cases}$$

where the initial equilibrium point is calculated with the *Equil_level_ice* function (cf. Level ice resistance, II.2.2.10). This step lasts for a user-defined time interval, $t_{init}$.

**A-B** In this first step of the ridge going process, the input delivered power is progressively increased from its initial level $P_D^0$ to its maximum value $P_D^{max}$. Considering that the engine is able to linearly increase its power from 0 to $P_D^{max}$ in a time interval $t_{0\to P_D^{max}}$, we have

$$P_D^{AB}(t) = P_D^0 + (t - t_A)\frac{P_D^{max}}{t_{0\to P_D^{max}}}$$

The thrust is then calculated with $Thrust\_calc$, and the resistance with $Ice\_resistance$ : as the power increases, the thrust increases and the ship starts to accelerate when $T_{net} > R_{ice}$. Note that this change in velocity induces a change in the ice resistance and in the thrust itself, so that these quantities have to be evaluated at each time step.

**B-C** Step $B-C$ starts when the delivered power reaches $P_D^{max}$. The ship continues to increase her velocity, while the ice resistance is increasing. If step $B-C$ was infinitely long, the ship would accelerate until she reaches $V^{max}$, the theoretical maximum velocity in level ice, computed with the method described in subsection II.2.2.10.

Practically, 95% of $V^{max}$ is considered to be satisfactory (each additional 1% of $V^{max}$ would require an excessively long time compared to the gain in kinetic energy) and point $C$ is reached when

$$\dot{x}^k > 0.95 \cdot V^{max}$$

**C-D** This step is purely parametrically defined. It simply corresponds to the distance the ship travels between point $C$ ($V$ reaching 95% of $V^{max}$) and point $D$, where the ship's bow hits the ridge. This step can be viewed as an extension of step $B-C$, but defined by a length :

$$L_{CD} = x_D - x_C$$

Typically, $L_{CD}$ is of the same order of magnitude as $L_{ship}$.

**D-E** In step $D-E$, the ship's bow hits the ridge, so that the resistance rapidly increases. The overall maximum resistance is assumed to occur when the fore shoulder attains the center of the ridge.

Calculation of the corresponding resistance requires rather long developments, given in Mellor [13], Ueland & Slettebø [7] and Ehle [8]. In order to keep clear structure and conciseness of the report, only the main formula used to compute this maximum resistance $R_{ridge}^{max}$ is given here. Further developments are given in Appendix B.

$$R_{ridge}^{max} = \frac{kv_{pen}}{\sqrt{gH_k}}(BR + 2\mu L_{bow}R) + 0.63\frac{kv_{pen}}{\sqrt{gH_k}}(2\mu RNL_{mid})$$

with

- $k$ and $N$ are empirical parameters, cf. ranges of variations given in Appendix;
- $v_{pen}$ the penetration velocity (i.e. the ship's velocity at the instant of bow impact);
- $H_k$ the ridge keel depth;
- $B$ is the ship's breadth;
- $R$ is the ice yield resistance

  $$R = \frac{1}{2}\frac{1+\sin\phi_s}{1-\sin\phi_s}(1-\eta)\rho_i g\left(1-\frac{\rho_i}{\rho_w}\right)H_k^2$$

  with $\rho_i$ and $\rho_w$ respectively the ice and water densities, $\phi_s$ the inner friction angle and $\eta$ the ice porosity (cf. Appendix for more details and typical values);
- $\mu$ is the friction coefficient of the ice on the hull paint;
- $L_{bow}$ and $L_{mid}$ are respectively the ship's bow length and parallel midbody length.

Finally, the resistance in step $D-E$ is calculated by a linear interpolation between $R_{ice}^D$ and $R_{ice}^E = R_{ridge}^{max}$ :

$$R_{ice}^k = R_{ice}^D + (x^k - x^D)\frac{R_{ice}^E - R_{ice}^D}{L_{DE}} \qquad L_{DE} = L_{bow} + \frac{1}{2}L_{ridge}$$

where $L_{ridge}$ is the ridge length and $L_{DE}$ the length of step $D-E$.

**E-F** The maximum ice resistance has been reached in E, and now the bow is leaving the ridge. During this step, the bow resistance decreases, while the midbody resistance increases. Altogether, the resistance is smoothly decreasing until the midbody resistance $R_{mid}$. This midbody resistance has been evaluated by Myland [8] as 63% of the maximum ridge resistance, with the condition to stay above the level ice resistance[15], so that

$$R_{ice}^{F} = \max(0.63 \cdot R_{ridge}^{max}, R_{level\,ice})$$

Between $E$ and $F$, the resistance is then modeled with a decreasing exponential function (Fig. 18).

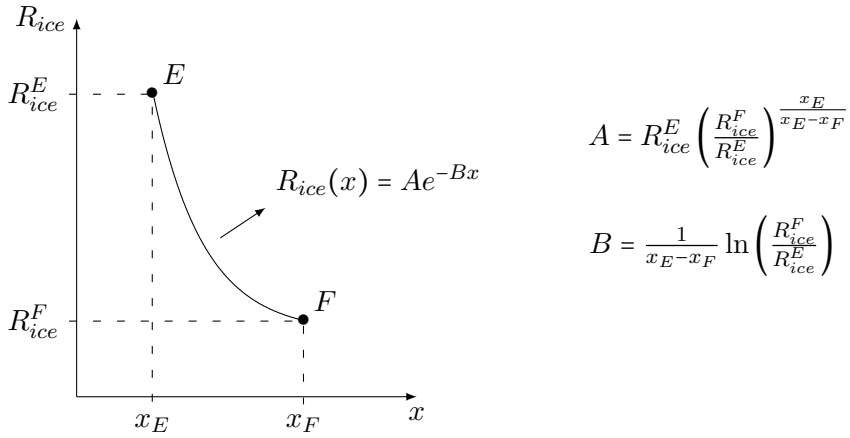



Figure 18: Step $E - F$ : Modeling of ice resistance

**F-G** During this next step, only the ship's parallel midbody is in contact with the ridge, so that the resistance is constant and still equal to

$$R_{ice}^{k} = R_{ice}^{F} = \max(0.63 \cdot R_{ridge}^{max}, R_{level\,ice})$$

This lasts until the stern reaches the ridge, so that $L_{FG} = L_{mid} - L_{ridge}$.

**G-H** The parallel midbody is then leaving the ridge, while the stern comes in contact. The midbody resistance thus decreases and the stern resistance increases (mainly because of the appendages), so that globally the resistance smoothly decreases until the level ice resistance.

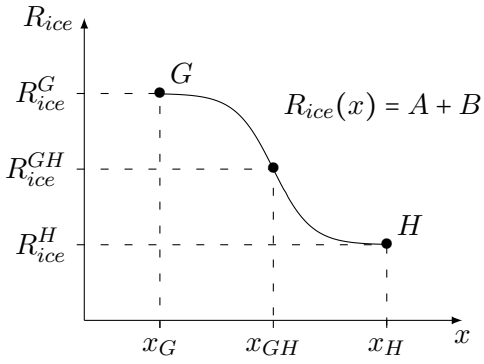
In this report, it has been decided to represent such a smooth variation by a hyperbolic tangent function, as described in Fig. 19, with $x_H = x_G + L_{ridge}$. The slope $m$ is then defined so that in $H$ the resistance reaches 101% of its asymptotic value $R_{level\,ice}^{H}$ :

$$m\,?\quad so\ that\quad R_{ice}^{H} = 1.01 \cdot R_{level\,ice}^{H}$$

so that

$$m = \frac{-1}{x_H - C} \operatorname{atanh}\left(\frac{1.01 \cdot R_{level\,ice}^{H} - A}{B}\right)$$

[15]This unexpected situation may occur when the ridges are relatively small compared to the level ice thickness.




$$R_{ice}(x) = A + B\tanh\left[-m(x-C)\right]$$

$$A = R_{ice}^{GH} = \frac{R_{ice}^{G} + R_{ice}^{H}}{2}$$

$$B = \frac{R_{ice}^{G} - R_{ice}^{H}}{2}$$

$$C = x_{GH} = \frac{x_G + x_H}{2}$$

Figure 19: Step $G-H$ : Modeling of ice resistance

However, at the beginning of step $G-H$, the level ice resistance in $H$ is unknown, as it depends on the ship's velocity in $H$. Therefore, an iterative process has to be used to determine the correct resistance curve :

1. As first guess, the maximum ice resistance, i.e. the ice resistance associated to the theoretical maximum velocity in level ice $V^{max}$, is used :

$$R_{ice}^{H*} = R_{level\,ice}^{max}$$

2. The step $G-H$ is then calculated according to the resistance curve in hyperbolic tangent.

3. When reaching point $H$, the corresponding velocity $\dot{x}^H$ is used to compute the level ice resistance according to Lindqvist (function $Ice\_resistance$), $R_{ice}^{H}$.

4. Then, the relative error between $R_{ice}^{H}$ and $R_{ice}^{H*}$ is evaluated :

$$Rel.Error = \frac{R_{ice}^{H} - R_{ice}^{H*}}{R_{ice}^{H}}$$

   If $Rel.Error < 1\%$, then the resistance curve is sufficiently accurate and we can jump to step $H-I$. If not, an iterative Newton-Raphson algorithm[16] is used to compute a new estimate $R_{ice}^{H*}$ and the process is repeated from 2. until convergence.

**H-I** In step $H-I$, the ship is back in level ice, but keeps full power in order to recover speed. Mathematically, this step is very similar to $C-D$, as its length is purely parametrically defined :

$$L_{HI} = x_I - x_H$$

Typically, $L_{HI}$ is of the same order of magnitude as $L_{CD}$, but mainly depends on the velocity loss that has occurred in the ridge.

[16] More details in the MatLab code given in Appendix D.

**I-J** The delivered power is then decreased back to its initial level $P_D^0$. Considering that the engine is able to linearly decrease its power from $P_D^{max}$ to 0 in a time interval $t_{P_D^{max}\to 0}$, we have

$$P_D^{AB}(t) = P_D^{max} - (t - t_I)\frac{P_D^{max}}{t_{P_D^{max}\to 0}}$$

**J-K** The power is now at its initial level $P_D^0$, and the velocity will increase or decrease progressively until it gets back to its initial value in level ice $V_0$. Practically, this step is assumed to be accomplished when the velocity reaches 95% of $V_0$ (if it was increasing) or 105% of $V_0$ (if it was decreasing). Point $K$ is then reached when the following general condition is satisfied :

$$\dot{x}^k > 0.95 \cdot V_0 \qquad and \qquad \dot{x}^k < 1.05 \cdot V_0$$

**K-End** Finally, step $K - End$ is optional and similar to $0 - A$ : the ship is in level ice in equilibrium conditions. Delivered power, thrust and ice resistance are at their initial level and acceleration is 0. The ship is in constant linear motion for a time $t_{end}$ defined by the user.

***Check for ramming***

As already explained previously, this code is not suitable when the ridge breaking process requires ramming operations. Therefore, a stop condition has been added to the MatLab script, so that the process is interrupted if the ship's velocity becomes negative :

$$if \quad \dot{x}^k < 0 \quad \rightarrow \quad Stop$$

This means that the ship is unable to break the ridge at the first advance, so that ramming is required.

### *2.4.4 Validation*

The method for estimating the ship motion through an ice ridge is now applied for some reference tests in order to validate the numerical simulation. In this subsection, the reference data will consist in experimental test results from model tests carried out at the Hamburg Ship Model Basin.

***Ship dimensions, ridge data and simulation parameters***

The tests were performed with the model ships n$^o$1 and n$^o$2 already used for the level ice resistance validation (cf. Table 1 in subsection II.2.2.11). The main ridge dimensions and ice properties related to each test are given in Table 2. Please refer to Appendix A.2 for the complete data set of parameters.

Table 2: Ridge breaking resistance : validation data

| Validation test | | A | B | C | D |
|---|---|---|---|---|---|
| Vessel | | Ship n$^o$2 | Ship n$^o$2 | Ship n$^o$2 | Ship n$^o$1 |
| $L_{ridge}$ | $[m]$ | 42.0 | 30.0 | 42.0 | 40 |
| $H_k$ | $[m]$ | 7.6 | 6.9 | 7.6 | 7.5 |
| $H_{ice}$ | $[m]$ | 0.9 | 0.95 | 0.9 | 1.25 |
| $H_{snow}$ | $[m]$ | 0 | 0 | 0 | 0 |
| $\sigma_b$ | $[kPa]$ | 1100 | 1000 | 1100 | 1450 |

***Remark about simulation parameters***

A remark must be done about the simulation parameters. We have seen that the calculation of the maximum ridge resistance involves the use of several empirical parameters. These parameters typically vary in the following ranges (cf. Appendix B for more details) :

$$k \in [1.1, 1.75] \qquad N \in [0.06, 0.50] \qquad \phi_s \in [42^o, 58^o]$$

In a first approach, the centre values were selected, but it appeared that the maximum ridge resistance $R_{ice}^E$ was relatively too small compared to the level ice resistance, even lower in some cases. The explanation comes from the ice bending strength used for the experimental tests. Its very high value (above $1000\,kPa$) is rather uncommon and quite different from the range most frequently observed in nature ($500 - 900\,kPa$).

Additionally, it can be remarked that the formulation of the ridge maximum resistance does not consider any influence of the ice bending strength, and therefore, the ridge resistance would be the same whatever the value of the bending strength. For this reason, values close to the upper limit of the parameter ranges were selected, and much better results have been observed, as it will be described below.

***Validation results***

The simulation results related to the four validation tests are compared to the experimental[17] measurements. Since tests $A$, $B$ and $C$ produce relatively similar conclusions, only results of tests $A$ and $D$ will be presented here. Please refer to the Appendix for tests $B$ and $C$.

In all validation tests, the simulation always starts with a ship initially at rest. Then, the delivered power is increased, so that the ship accelerates and hits the ridge. After the ridge is passed, power is decreased to 0 and the simulation ends shortly after the ship reaches zero velocity again.

[17]During each experimental test, model scale data have been recorded. The full scale data have then been recovered with the scaling rules of the Froude similitude (cf. Appendix A.2.3).

**Test A (Fig. 20)** In this first test, the delivered power starts to increase in $t = 75\,s$. As initially, ice resistance is larger than propeller thrust, the ship starts to move only in $t \approx 85\,s$. This non-zero ice resistance at zero speed is however not observed in the experimental data, as the sensor was onboard the ship. The velocity then increases until the equilibrium velocity in level ice ($C - D$), that is relatively close for the experiment and the simulation.

The maximum resistance in the ridge (point $E$) appears to be less sharp in the experiment, but the maximum points are close in both cases. However, when the ship is leaving the ridge (point $F$ to $H$), the ice resistance keeps a very high value in the experiment, which could not be modeled in the simulation, as the midbody resistance $R_{mid}$ was already lower than the level ice resistance.

This good modeling until point $E$ and then less good after is also observed on the motion diagram : the position and velocity are relatively well modeled until point $E$, but the experiment shows a bigger velocity drop in the ridge (the ship almost gets stuck). However, it is worth to notice that the acceleration is correctly modeled during the whole process.

Similar conclusions are obtained for validation tests $B$ and $C$ (cf. Appendix C).

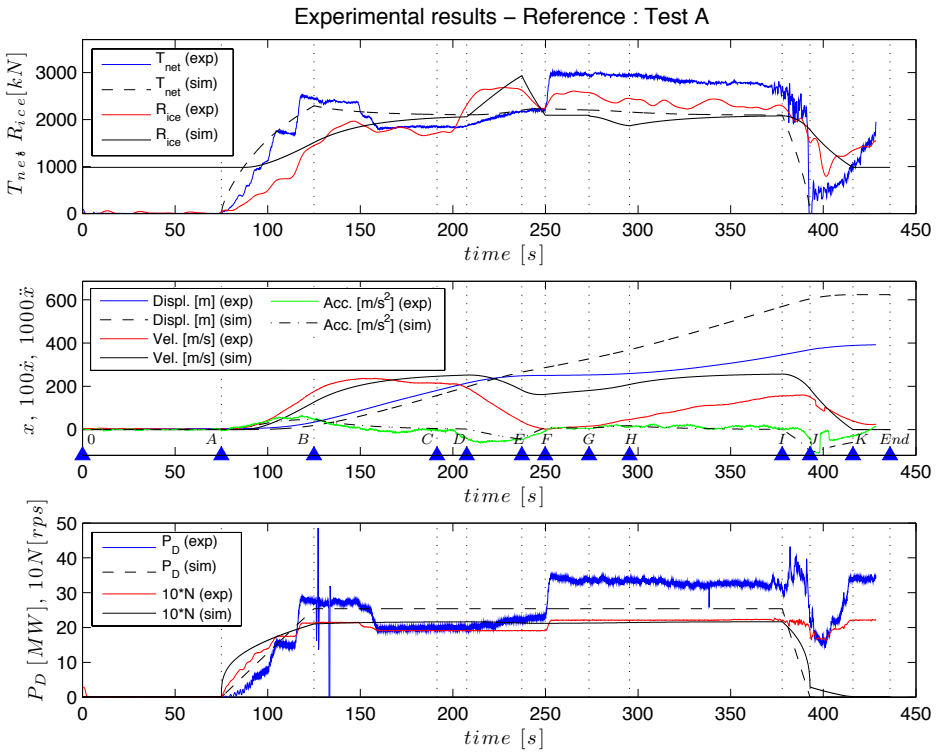



Figure 20: Ridge breaking validation : test A

**Test D (Fig. 21)** Test $D$ is an example of limitation for the simulation model. The ice bending strength is so high ($1450\,kPa$) that the simulation cannot model correctly the maximum resistance of the ridge (point E), even with the upper-range values for $k$, $N$ and $\phi_s$. This shows that the model requires some additional studies to take into account the influence of the ice bending strength on the ridge resistance.

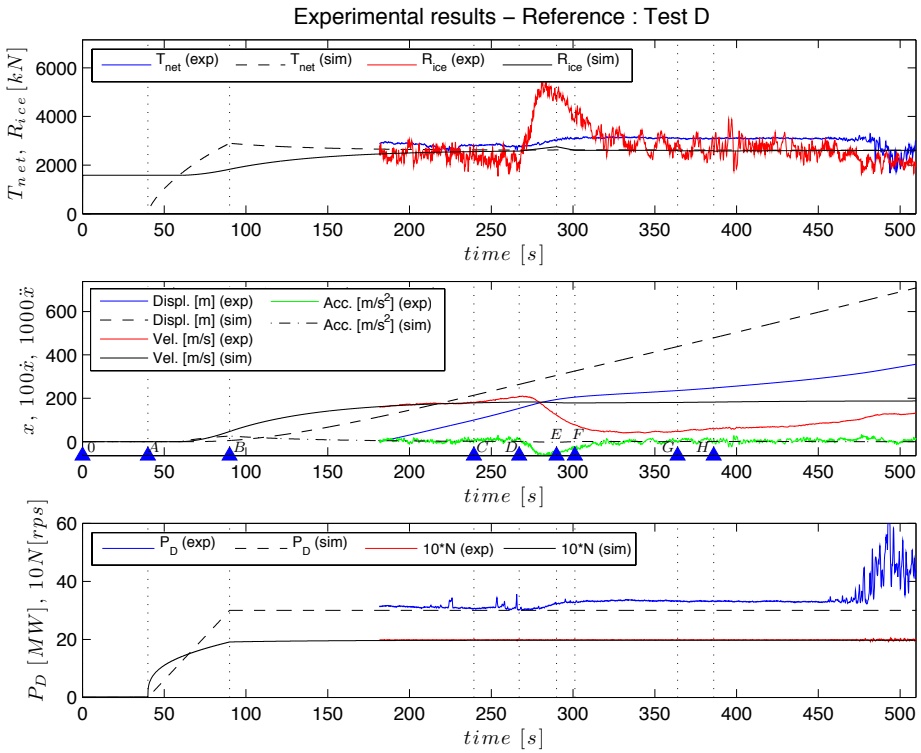



Figure 21: Ridge breaking validation : test D

**General comments and conclusions about the validation process** If we put aside the example of test run $D$ for which the ice bending strength was very high, and considering the complexity of the ridge resistance modeling, the simulation performance is quite satisfactory. The acceleration and deceleration phases are correctly modeled, the maximum velocity in level ice is reached quite similarly to the experimental tests, and the maximum ridge resistance is not so different in both cases.

However, some drawbacks and uncertainties must be brought to light. If the time required to overcome the ridge is quite similar in simulation and experiments, the total displacement of the ship is always overestimated by more than 50%, and the velocity drop is not always correctly modeled. Moreover, the validation tests were all considering high strength ice, so that the midbody resistance did not appear in the results (it was lower than level ice resistance). It would therefore be useful to validate the model with data from experimental tests in softer ice.

In a more general way, it is important to remark that the simulation results given in this section have been obtained by selecting the appropriate values of the parameters $k$, $N$ and $\phi_s$ only *after* comparison with the experimental results. It would have been more difficult to guess these values without any reference data. Some additional studies seem to be required to determine how the ice bending strength, but also the ridge geometry and probably the bow geometry may influence the ridge resistance.

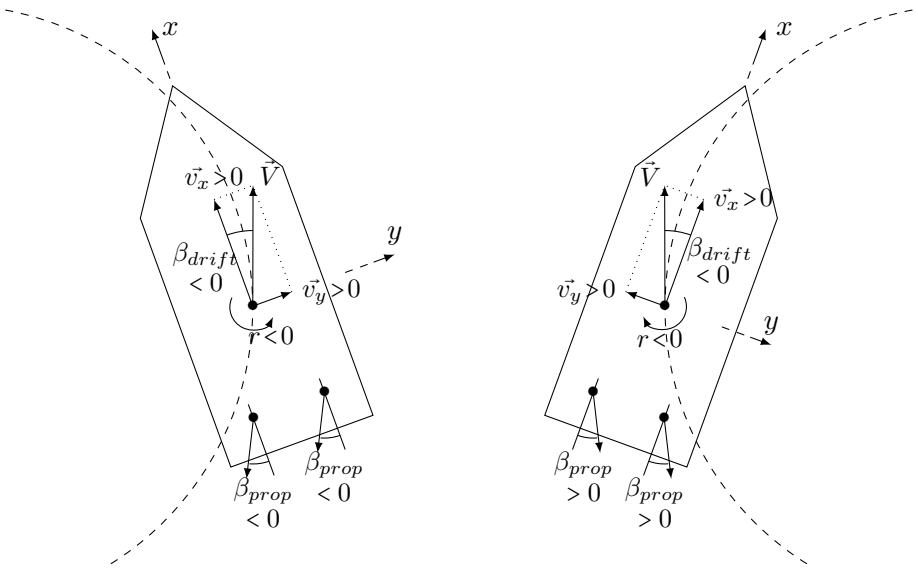
## 2.5 Manoeuvrability Estimation

After calculation of the resistance parameters in linear motion, another critical point of the simulation is the estimation of the vessel manoeuvrability. This section is dedicated to the estimation of the manoeuvering parameters requested for a ship to execute a predefined turn : power level, turning velocity, pods/rudder angle, ...

In the following, the calculation method is presented, with details about the determination of the ice resistance in turning motion, calculation of the propeller(s) thrust and rudder forces. Similarly to the previous sections, a validation of the simulation results is performed at the end of this section.

### *2.5.1 Calculation Method*

In the present report, we are looking for a quite simple estimation of the turning simulation, so that only the steady turning parameters will be calculated[18]. The steady turning configuration is represented on the drawings of Fig. 22, with indications of the signs conventions that are considered. Note that the figure is related to a double pod-propelled ship, but the software is capable of simulation for any number of propellers, and also for rudder and fixed propeller configuration (cf. II.2.5.4).




(a) Turning to port (b) Turning to starboard

Figure 22: Steady turning configuration

According to the management technique that is selected (cf. section II.3), one of the two following problems must be solved :

*Calc_circle_V.m* :

$$\left.\begin{array}{l}\text{Turning velocity}\\ \text{Turning radius}\end{array}\right\} \quad given \quad \Rightarrow \quad \left\{\begin{array}{ll}\text{Turning Power} & \\ \text{Pods/rudder angle} & unknown\\ \text{Drift angle} & \end{array}\right.$$

[18]This is quite reasonable, as for ice management simulation, only the mean velocity or the mean requested power during a pre-defined turning manoeuvre is to be determined. The exact vessel trajectory, or the time series of the pods angle is not requested.

*Calc_circle_P_D.m* :

$$\left.\begin{array}{l}\text{Turning power}\\ \text{Turning radius}\end{array}\right\} \; given \quad \Rightarrow \quad \left\{\begin{array}{ll}\text{Turning velocity} & \\ \text{Pods/rudder angle} & unknown \\ \text{Drift angle} & \end{array}\right.$$

We are then looking for a valid[19] combination of the 3 unknowns such that the system is in equilibrium, i.e. the total forces along $x$ and $y$ and the total yaw moment are all zero :

$$\sum F_{x,y} = 0 \qquad and \qquad \sum M = 0$$

A simple sketch of the various forces acting on the ship in turning motion is given in Fig. 23. These forces will be divided into several smaller components and then estimated in the next subsections of this chapter.

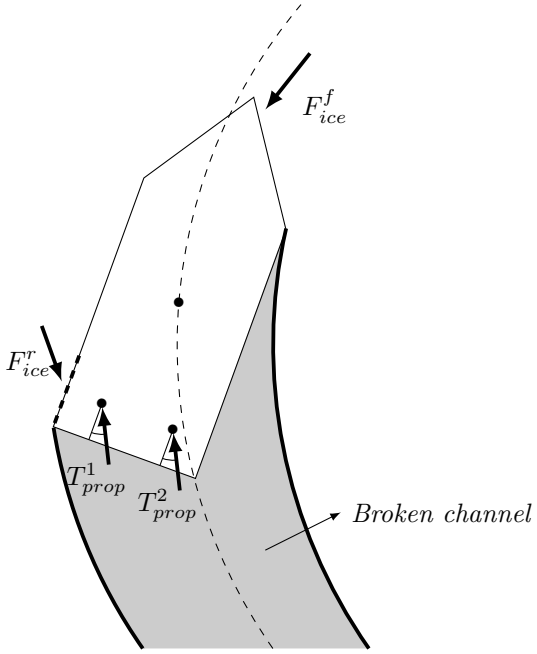



Figure 23: Forces acting on the ship in turning motion

### *2.5.2 Ice Resistance on the Fore Part*

In order to estimate the ice resistance on the fore part, the Lindqvist formulation [6] for ice resistance in linear motion has been improved to take into account the turning trajectory.

When the ship is in turning motion, the velocity of the incoming ice at the bow is different from the velocity $V$ of the centre of gravity, for the two following reasons :

- a drift angle $\beta_{drift}$ is present between the forward velocity $\vec{v}_x$ of the ship in its local coordinate system, and the actual velocity vector $\vec{V}$, so that $\vec{v}_x \neq \vec{V}$;
- due to the yaw velocity $r$, the velocity of the incoming ice depends on the distance to the ship's centre of gravity.

[19] As the problem is nonlinear, several combinations of the 3 unknowns may result in a valid turning configuration. The final solution must then be selected among the various resulting valid solutions (cf. II.2.5.6).

For these reasons, the Lindqvist method is applied as described in section II.2.2, but with significative changes[20] :

- The various components of the ship resistance (bending at the bow, crushing at the stem, potential energy loss, frictional resistance, wedge resistance and open-water resistance) are computed independently, so that they all have a different direction, amplitude and application point.

- For each component $j$, a *modified incoming velocity* $V^j$ is used, and therefore a *local drift angle* $\beta^j$:

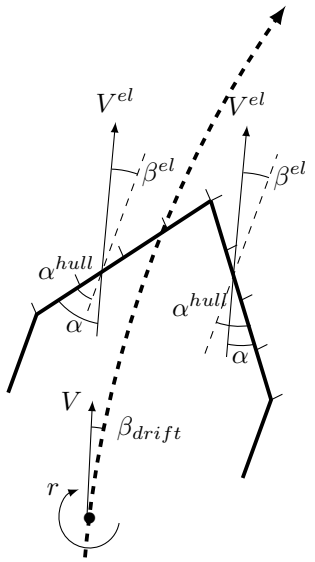
$$V^j = \sqrt{(V_x^j)^2 + (V_y^j)^2} \quad with \quad \begin{aligned} V_x^j &= V_x^{ship} \\ V_y^j &= V_y^{ship} + r \cdot x_c \\ \beta^j &= -\arctan \frac{V_y^j}{V_x^j} \end{aligned}$$

  where $x_c$ is the distance along $x$ of the application point to centre of gravity.

- Each resistance component is also estimated with *modified hull angles*, related to the actual direction of the incoming ice :

$$\begin{aligned} \psi &= \psi^{hull} \\ \alpha &= \alpha^{hull} + \beta^j \\ \phi &= \arctan\left(\tan\psi \sin\alpha\right) \end{aligned}$$

- The bending resistance is now computed on 8 elements along the ship beam (4 on each side of the centreline), so that each element is submitted to a different resistance force, with a different lever arm regards to the centre of gravity. Note that in this case, the modification on the local angle $\alpha$ depends on the side regards to the centreline, as explained in Fig. 24.




Inside the turn :

$$\alpha = \alpha^{hull} - \beta^{el}$$

Outside the turn :

$$\alpha = \alpha^{hull} + \beta^{el}$$

Figure 24: Calculation of modified hull angles at the bow

[20] Additionally, in the corresponding MatLab code, a parameter *turn_sign* has been used to satisfy a few signs conventions when the ship is turning to port (+1) or to starboard (−1). For ease of reading, this parameter has been omitted in the equations of the present report. Any interested reader is invited to refer to the code given in Appendix D.

In order to avoid many repetitions with the formulas given in the level ice resistance section (cf. II.2.2), only the main elements used for the resistance calculation of each component are given here, in Table 3.

Table 3: Calculation of the ice resistance at the fore part, according to the modified Lindqvist formulation

| Component | Application point (distance from CG) | Velocity influence $R^j \times \ldots$ |
|---|---|---|
| Crushing at the stem | *Stem* | $1 + 1.4\frac{V_{stem}}{\sqrt{g \cdot H_{ice}}}$ |
| Bending at the bow (8 elements) | *Centre of the element (on the waterline)* | $1 + 1.4\frac{V^{el}}{\sqrt{g \cdot H_{ice}}}$ |
| Potential energy loss | *Centre of bow and ice-covered bottom areas* | $1 + 9.4\frac{V_p}{\sqrt{g \cdot L_{pp}}}$ |
| Frictional resistance | *Centre of bow and ice-covered bottom areas* | $1 + 9.4\frac{V_{fr}}{\sqrt{g \cdot L_{pp}}}$ |
| Wedge resistance | *Stem* | $V_{stem}^2$ |
| Open-water resistance | *Stem* | $\frac{2}{3}$ |

### *2.5.3 Ice Resistance on the Aft*

When the ship is turning, compared to linear motion, a new resistance component must be added at the aft part of the ship, to take into account the crushing and frictional resistance that appears when the aft exerts a force on the ice (Fig. 25).

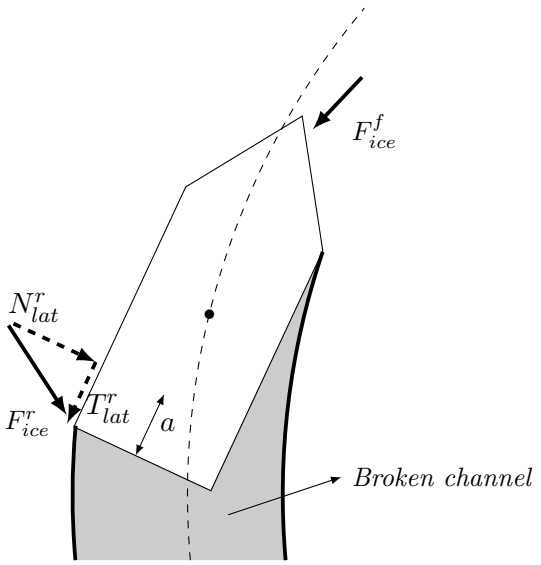



Figure 25: Ice resistance on the aft

However, this resistance can hardly be estimated with the Lindqvist formulation, as the hull geometry at the aft is generally so that $\psi \approx 90^o$, which implies that the crushing and bending resistances are close to infinity. Instead, the use an empirical parameter $k_{turning}$ to estimate

this resistance is preferred. The normal force on the hull will be proportional to the total ice resistance on the fore part :

$$N_{lat}^{r} = k_{turning} \cdot F_{ice}^{f} \cdot \frac{L_{pp}}{R_{turning}}$$

where $F_{ice}^{f}$ is the sum of all the resistance components on the fore part described in Table 3. In the previous expression, the factor $\frac{L_{pp}}{R_{turning}}$ is used to make the normal force proportional to the ship's length and inversely proportional to the turning radius. The lateral force is then obtained from the friction coefficient $\mu$ of the ice on the hull paint :

$$T_{lat}^{r} = -\mu \cdot N_{lat}^{r}$$

and this force is assumed to act on a length $a = \frac{L_{pp}}{4}$ from the aft perpendicular of the ship, so that the lever arms around the CG are respectively $x_{cg} - a/2$ for $N_{lat}^{r}$ and $B/2$ for $T_{lat}^{r}$ (with $x_{cg}$ measured from the aft perpendicular).

A first subsection of the validation process (cf. II.2.5.7) will be dedicated to the determination of the $k_{turning}$ coefficient.

### *2.5.4 Propeller(s) Thrust*

The thrust of each propeller is then calculated from the open-water characteristics, with the function

$$[T_{net},\ N_{calc}] = Thrust\_calc\_1prop\,(P_D,\ V,\ Open-water\,characteristics)$$

This function is very similar to the *Thrust_calc* function described in section II.2.1, but for only one propeller, as the thrust of each propeller is computed independently (specific incoming velocity).

Then, a specific calculation is performed depending on the propulsion system.

**Pods** When the ship is equipped with pods, or azimuth thrusters, the total net thrust along $x$ and $y$ is simply calculated by projection on the axes :

$$T_{x}^{tot} = \sum_{i} T_{prop}^{i} \cos\beta_{prop}^{i} \qquad T_{y}^{tot} = -\sum_{i} T_{prop}^{i} \sin\beta_{prop}^{i}$$

where the minus sign for $T_{y}^{tot}$ is due to the orientation of the coordinate system (Fig. 22).

Additionally, it is worth to notice that the software has been written so that three pods configurations are possible during the turning motion :

| Mode | Description |
|---|---|
| 1 | All pods are rotating |
| 2 | Two pods, only the pod outside turn rotates |
| 3 | Two pods, only the pod inside turn rotates |

**Rudder** When the ship is equipped with a fixed propeller and rudder, it is necessary to evaluate the resulting force acting on the ship and the corresponding application point (Fig. 26). For this purpose, the MatLab simulation uses an executable software developed by the Hamburg Ship Model Basin. From

- the rudder geometry, position, lift, drag and moment coefficients, and
- the propeller geometry, dimensions, and open-water characteristics,

and for a given ship velocity, propeller revolution rate and rudder angle, the executable computes the resulting forces[21] along $x$ and $y$, and the position of the application point on the centreline (Fig. 27). Examples of input data files are given in Appendix E.

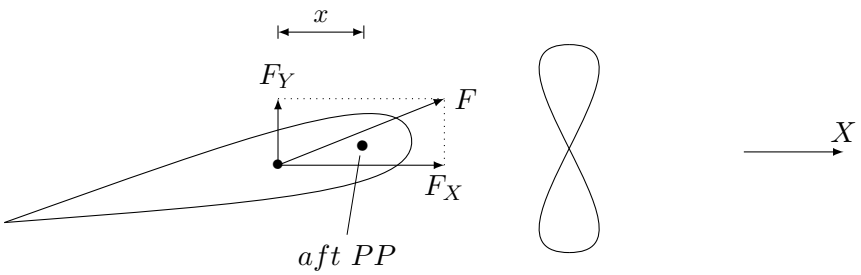



Figure 26: Calculation of total rudder forces

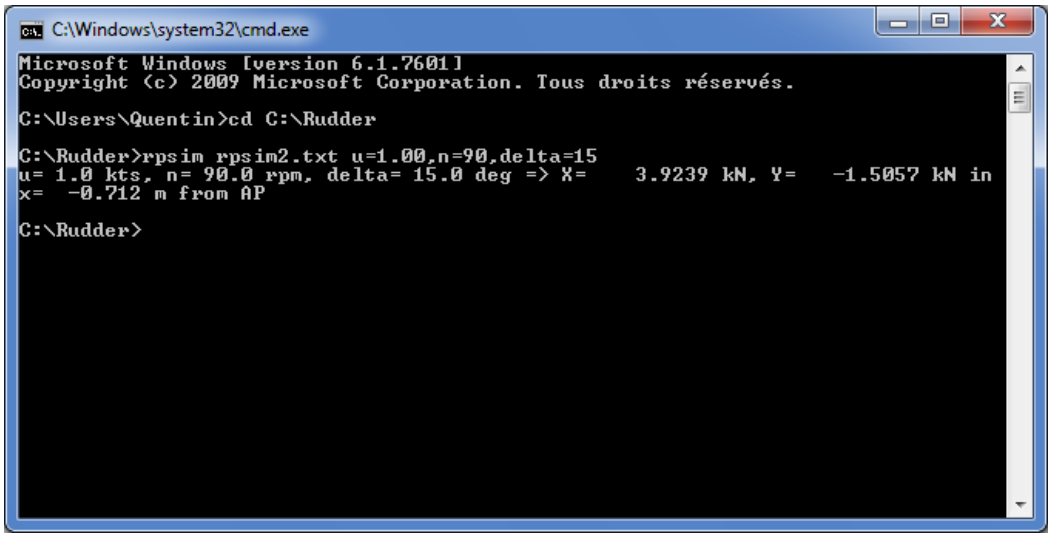



Figure 27: Executable for rudder forces calculation

These operations are performed by the *Rudder_force_map* function, that runs the executable and reads the outputs.

[21] The force along the $x$-axis is the resultant of propeller thrust and drag force.

### *2.5.5 Total Forces and Moment*

**Forces** Finally, the total forces along $x$ and $y$ acting on the ship are calculated by summing the contributions of the forces components :

$$F_x^{tot} = \sum_i (F_{ice}^f)_i \cos\beta_i + T_{lat}^r + T_x^{tot}$$

$$F_y^{tot} = \sum_i (F_{ice}^f)_i \sin\beta_i + N_{lat}^r + T_y^{tot}$$

where, in this expression, the ice resistance components at the fore part $(F_{ice}^f)_i$ are considered to act in the direction of the local velocity.

**Moment** The contribution of each force vector to the global yaw moment can be determined from the product of the force amplitude $F_i$ by the lever arm $l_i$ regards to the centre of gravity :

$$M_i = F_i \cdot l_i$$

The expression of the lever arm is obtained from the drawing of Fig. 28. In the $yox$ coordinate system, the straight line $d$ has a slope $\frac{1}{\tan\beta}$, so that its equation can be found to be

$$d \equiv x = -(\cot\beta)y + x_P + (\cot\beta)y_P$$

while straight line $d'$, perpendicular to $d$ and passing through $O$ has the equation

$$d' \equiv x = \frac{1}{\cot\beta} y$$

Coordinates of point $A$ are then obtained from the intersection of $d$ and $d'$. We get

$$y_A = \frac{x_P + (\cot\beta)y_P}{\frac{1}{\cot\beta} + \cot\beta} \quad and \quad x_A = \frac{1}{\cot\beta} y_A$$

and the lever arm $l$ is then[22]

$$l = |OA| = \sqrt{x_A^2 + y_A^2}$$

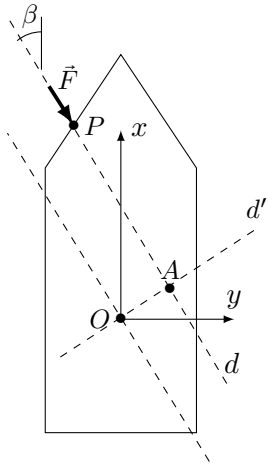



Figure 28: Calculation of the lever arm

[22]The particular case $\beta = 0$ is considered separately in the code, directly taking $l = y_P$.

### *2.5.6 Calculation*

The method described in the previous subsections is applied in the *Turning_circle* function, that gives as output the value of the total forces and yaw moment acting on the ship. Then, the computation of the turning configuration is performed in the *Calc_circle_V* or *Calc_circle_P_D* functions, depending on the requested parameter ($P_D$ or $V$).

The `fsolve` Matlab function is then used to determine a valid configuration for the turning. However, as the problem is highly nonlinear, it appears to have more than one solution, typically for two very different values of the drift angle. The Matlab solver converges towards one of the solutions depending on the value of the initial calculation point. In order to overcome this problem, it has been decided to run the solver for 9 different initial points, combinations of

*Calc_circle_V.m* :

$$P_D^0 = [1,\ 5,\ 10] \cdot P_D^{lin} \qquad \beta_{prop}^0 = [1,\ 25\ ,75]\, deg$$

*Calc_circle_P_D.m* :

$$V^0 = [0.1,\ 0.5,\ 1] \cdot V^{lin} \qquad \beta_{prop}^0 = [1,\ 25\ ,75]\, deg$$

where $P_D^{lin}$ and $V^{lin}$ are respectively the requested power and the equilibrium velocity for a linear motion in the same conditions. The initial drift angle is always $5^o$. The valid solution are then filtered out according to the following rules :

- the solver must have converged
- the pod(s)/rudder angle should be $< 90^o$
- the requested power level must be above $P_D^{lin}$ (for $Calc\_circle\_V$), or the calculated turning velocity must be positive (for $Calc\_circle\_P\_D$)

And the final solution is then selected by taking the one with the lowest requested power level (for $Calc\_circle\_V$) or with the highest turning velocity (for $Calc\_circle\_P\_D$) among the valid solutions.

### *2.5.7 Validation*

The calculation method described above can now be validated through comparison with experimental results. For this purpose, data from model tests performed at the Hamburg Ship Model Basin have been used. In this section, experimental data from three different ships are considered; their relevant dimensions are given in Table 4 (cf. Appendix A.3 for complete validation data) :

- the two first vessels are middle-size icebreakers used for research or emergency purposes. Their dual pods propulsion system allows a quite good manoeuvrability in ice and open-water.
- the last vessel is a small icebreaker for inland navigation, equipped with a single propeller and rudder configuration, quite common on smaller icebreaking vessels even if, regardless

of the cost and technical constraints, a multiple pods system would allow better manoeuvrability and give additional features to the icebreaker, like astern motion with propeller washing.

The testing conditions are also detailed in Table 4.

Table 4: Manoeuvrability estimation : validation data

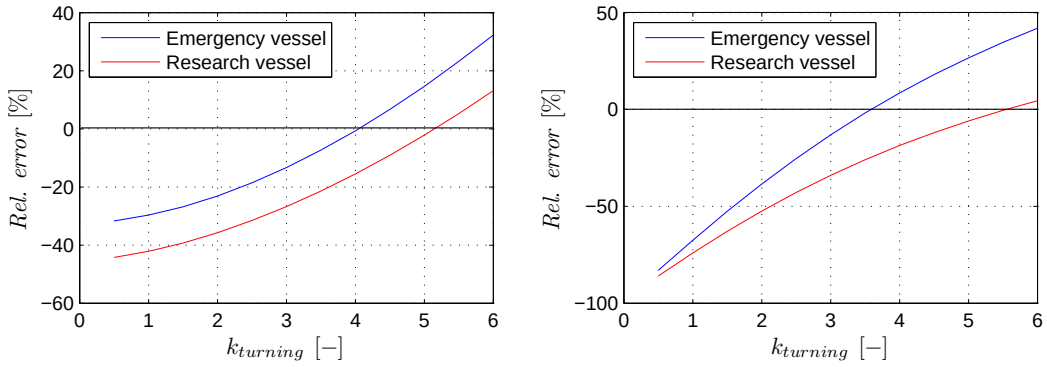
| Vessel | | Research vessel | Emergency vessel | Inland icebreaker |
|---|---|---|---|---|
| $L_{pp}$ | $[m]$ | 85.9 | 72.6 | 32.6 |
| $B$ | $[m]$ | 21.6 | 18.4 | 8.03 |
| $T$ | $[m]$ | 8.0 | 6.52 | 1.60 |
| $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ | $[deg]$ | [25.7, 33.6, 43.1] | [26.2, 36.6, 40.3] | [21.2, 21, 50.0] |
| $m_{ship}$ | $[ton]$ | 9850 | 6027 | 233.2 |
| *Propulsion* | [–] | Dual pod, $\phi = 4\,m$ $5\,m$ from centreline | Dual pod, $\phi = 3.5\,m$ $4.5\,m$ from centreline | Single screw $\phi = 1.5\,m$ + Rudder |
| $H_{ice}$ | $[m]$ | $1.0\,m$ | $0.50\,m$ | $0.30\,m$ |
| $H_{snow}$ | $[m]$ | $0.2\,m$ | $0.1\,m$ | $0.06\,m$ |
| $\sigma_b$ | $[kPa]$ | $500\,kPa$ | $500\,kPa$ | $500\,kPa$ |
| $R_{turning}$ | $[m]$ | $371\,m$ | $335\,m$ | $186\,m$ |
| $V_{turning}$ | $[m/s]$ | $0.74\,m/s$ | $0.88\,m/s$ | $1.02\,m/s$ |

**Influence of the $k_{turning}$ coefficient** First, the estimation of the $k_{turning}$ empirical coefficient used in the simulation is considered. In order to set good values for this coefficient, the simulation for the requested turning radius and turning velocity (cf. Table 4) has been run and compared with the experimental results.

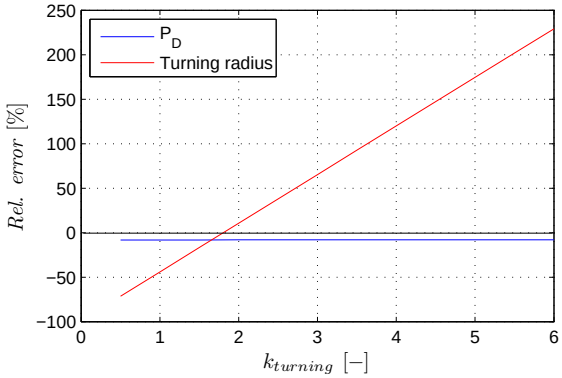
Fig. 29 shows the evolution of the relative error on the delivered power and the pods/rudder angle, as a function of the $k_{turning}$ coefficient. For the two first vessels, the simulation was run by imposing a value of the turning radius, and looking for the required delivered power and pods angle, as already explained previously. However, for the inland icebreaker, due to the very high experimental value of the rudder angle ($45^o$), the simulation code was not able to converge[23], so that the rudder angle has been set to $45^o$, and the solver was looking for the turning radius.

From the diagrams of Fig. 29, it can be deduced that $k_{turning}$ is generally in the range of 1.5 to 6. A more accurate approximation, and a link with the icebreaker size, purpose, turning velocity or turning radius would however require a deeper study of the simulation, with a large number of validation vessels. At this point, in order to estimate the manoeuverability of a given vessel, it is recommended to first determine the $k_{turning}$ parameter for a similar ship for which experimental data is available. This is a quite easy operation for the Hamburg Ship Model Basin, that can use its database of experimental tests to get a quite accurate simulation.

[23]The calculation worked only for rudder angles below $20^o$. Further coding has to be done in order to improve the convergence at higher rudder angles.




(a) Research and emergency vessels




(b) Inland icebreaker

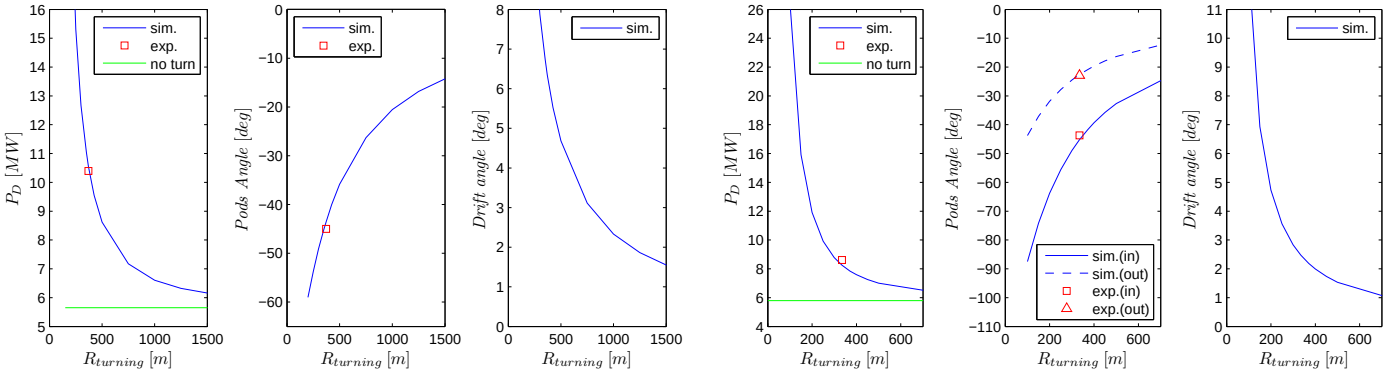
Figure 29: Influence of the $k_{turning}$ empirical coefficient

**Results** The results of the simulation are given in Fig. 30. The experimental points are plotted on the same diagrams as the simulation curves, for varying turning radius (research and emergency vessels) or varying rudder angle (inland icebreaker). Note that we have used for each simulation an appropriated value of $k_{turning}$, so that the simulation results are logically close to the experimental points.

Another important validation result to consider is the asymptotic behavior of the curves, i.e. when the turning radius increases, the delivered power converges towards the level ice power level, while the pods/rudder angles and the drift angle converge towards zero.

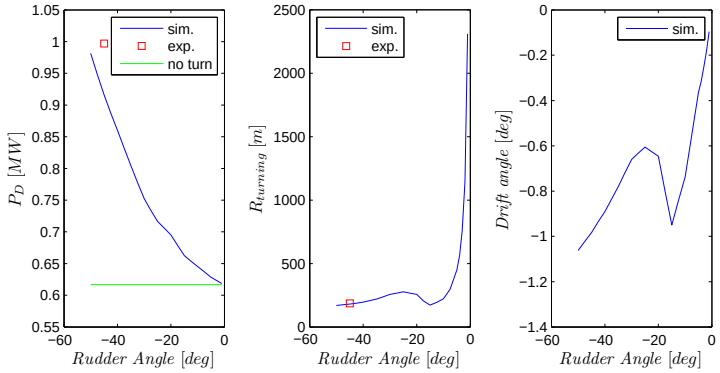
A last comment can be made about the simulation curves obtained for the inland icebreaker. We observe an unexpected behavior between $-15^o$ and $-40^o$ of rudder angle. The explanation comes from the lift and drag coefficients of the rudder profile (Fig. 31). Compared to a podded ship, where the thrust can be oriented in any direction without significant loss, the lift forces of the rudder significantly drops beyond the stall angle, so that the achievable turning radius increases (between $-15^o$ and $-25^o$ on the example of Fig. 30c). It finally goes back to a decreasing curve when the rudder angle further decreases.

The main element to remember about this simulation is that, for a given vessel, accurate results can only be obtained after fitting a good value for the empirical parameter $k_{turning}$ through comparison with experimental results for similar ship(s).




(a) Research vessel ($R_{turning}$ = 371 $m$, $V_{turning}$ = 0.74 $m/s$, $k_{turning}$ = 5.25)

(b) Emergency vessel ($R_{turning}$ = 335 $m$, $V_{turning}$ = 0.88 $m/s$, $k_{turning}$ = 3.75)




(c) Inland icebreaker ($R_{turning}$ = 186 $m$, $V_{turning}$ = 1.02 $m/s$, $k_{turning}$ = 1.75)

Figure 30: Validation results

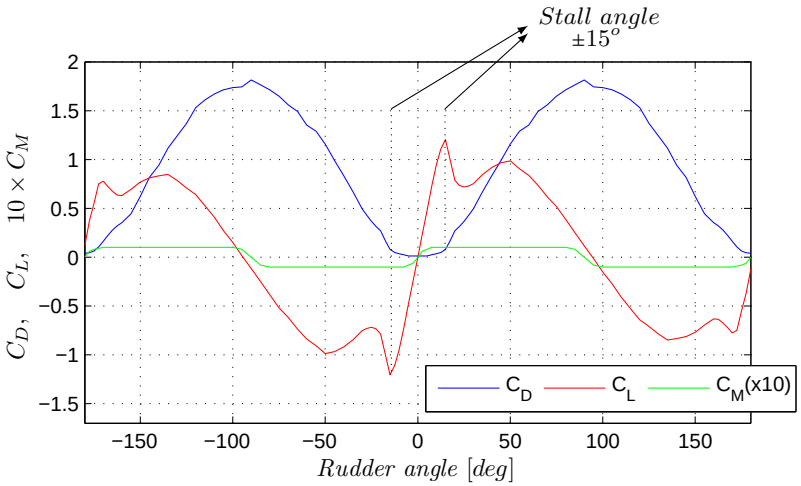



Figure 31: Lift, drag and moment coefficients of the rudder profile of the inland icebreaker

# 3 ICE MANAGEMENT STRATEGY AND SIMULATION

## 3.1 Generalities

### *3.1.1 Ice Management Configuration*

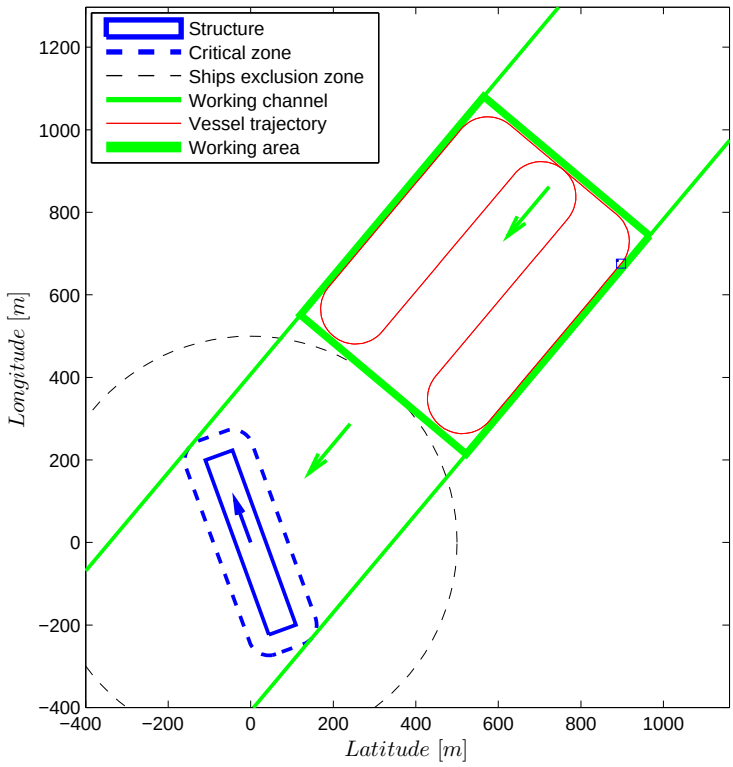
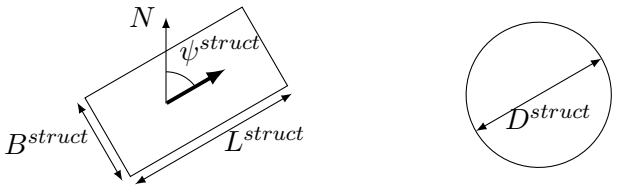


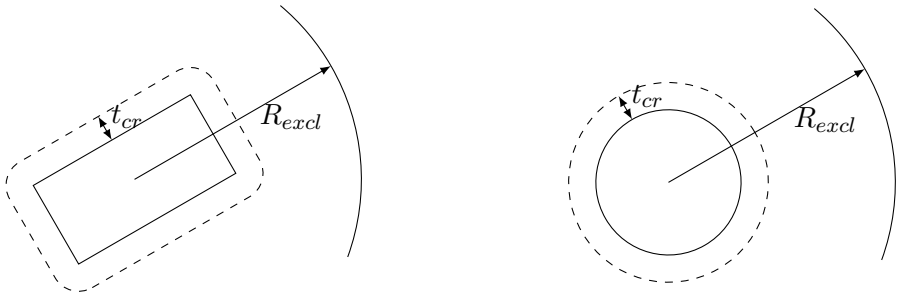

Figure 32: General presentation of the elements

The general configuration of typical ice management simulation is presented in Fig. 32. The following elements must be distinguished :

- The *structure* itself, that has to be protected from the drifting ice. The software has been written so that it can be rectangular or circular, with any orientation angle $\psi^{struct}$ :




- Around this structure, for safety reasons, there exists a *critical distance* $t_{cr}$ defining an area inside which no floes bigger than the target should enter. Additionally, the platform owner will also set an *exclusion radius* $R_{excl}$ defining an area inside which no ice management vessel can enter, so that the ice management area should be at a certain distance $R_{excl}$ from the platform centre :

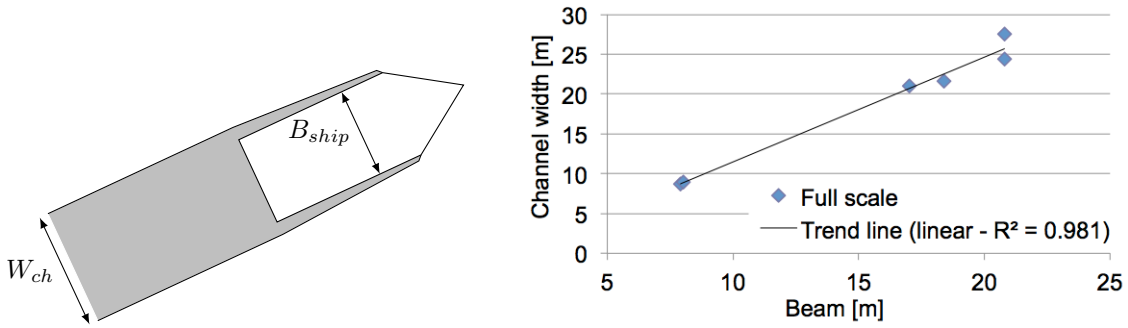



– According to the structure dimensions, orientation and the ice drift direction, the *working channel* is then defined. Along this channel, all the level ice will be managed, i.e. will be cut in small floes.

– Then is defined the *working area* (or *managed area*), inside which the icebreaker will work along a pre-defined *trajectory*. The dimensions of these characteristics will be defined in the sections 3.2 to 3.5 of this part of the report. Each of these sections is dedicated to a specific ice management technique.

### *3.1.2 Width of the Broken Channel*

When an icebreaker is moving in ice, it creates a broken channel, generally larger than its beam (Fig. 33a). In order to evaluate the width of this broken channel, it has been decided to empirically determine the ratio between the ship's beam $B_{ship}$ and the channel width $W_{ch}$, considering experimental data from the Hamburg Ship Model Basin.




(a) Width of the broken channel

(b) $W_{ch}$ *vs* $B_{ship}$

Figure 33: Broken channel

In Fig. 33b, the values of $B_{ship}$ and $W_{ch}$ are given for six different icebreakers (cf. Appendix A.4 for numerical values). This set of data can be confidently approximated by a linear trend line. As a result, in the following sections, the channel width is considered to be approximatively 130% of the ship's beam.

### *3.1.3 Floe Breaking*

In Fig. 32, it has been seen that, here in the case of a linear management technique, the ice-breaker cuts the ice in stripes, and not in square floes. This actually comes from a natural breaking of the floes. It is assumed, and this is experimentally verified, that when the ice management vessel cuts the ice, the stripes naturally break into more or less square shapes (Fig. 34).

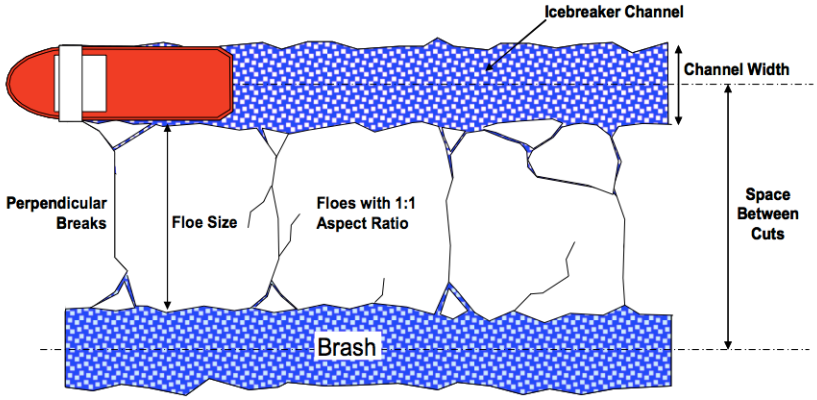



Figure 34: Breaking characteristics of ice floes [14]

As a result, in the following simulations of ice management operations, we will always look for trajectories that induce generation of stripes instead of square floes.

## 3.2 Linear Technique

The linear technique is the first ice management method described in the present report. As already shown in Fig. 32, the trajectory mainly consists of longitudinal passes parallel to the drift direction. At the end of each pass, the ice management vessel makes a $180^o$ turn and starts the next pass. Finally, at the end of the last pass, the vessel comes back to its initial position and starts a new management cycle.

Practically, this technique is used when ice drift speed is high but drift direction rather constant, as it is quite difficult to adapt the trajectory to a sudden change in drift direction.

### *3.2.1 Management Passes*

Once the drift direction is known and the structure dimensions and orientation defined, the width of the managed area can be defined. According to the drawings of Fig. 35 (circular structure) and Fig. 36 (rectangular structure),

$$W_{area} = \begin{cases} D^{struct} + 2t_{cr} & \textit{if circular structure} \\ \left|L^{struct}\sin(\alpha_{drift} - \psi^{struct})\right| + \left|B^{struct}\cos(\alpha_{drift} - \psi^{struct})\right| + 2t_{cr} & \textit{if rectangular structure} \end{cases}$$

and the required width of the trajectory of the management vessel is then

$$W_{traj} = W_{area} - W_{ch}$$

Once the target floe size $d_{floe}^{target}$ is defined (input of the simulation), the number of linear passes that are required to cover the management area is calculated (Fig. 37a) :

$$N_{pass} = ceil\left(\frac{W_{traj}}{W_{ch} + d_{floe}^{target}}\right) + 1$$

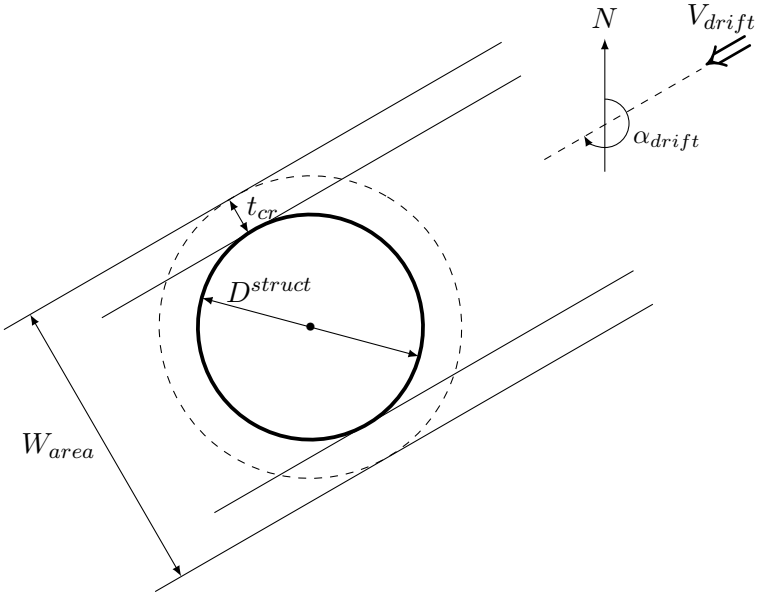



Figure 35: Linear technique : calculation of the width of the managed area for a circular structure

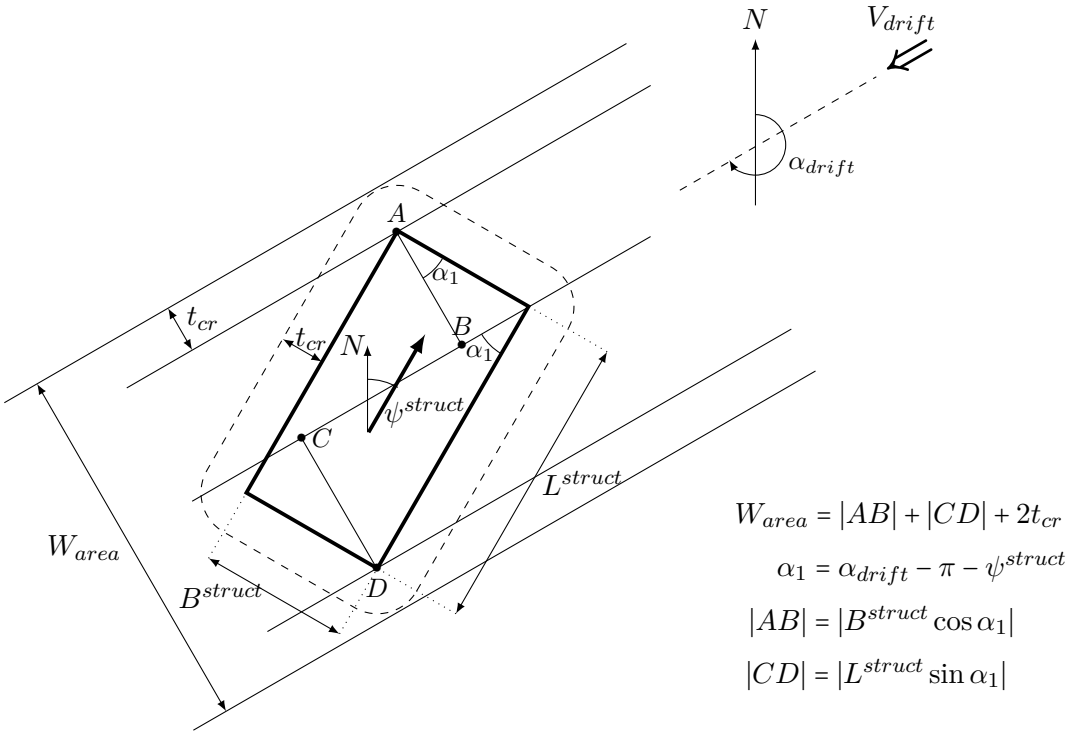
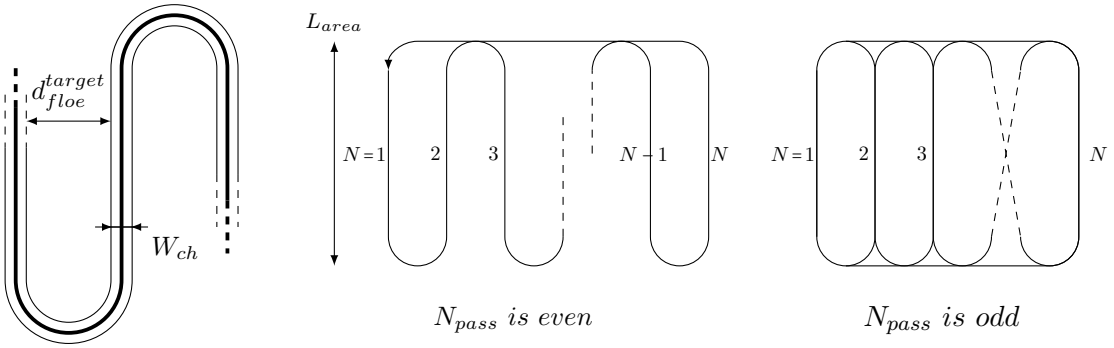



Figure 36: Linear technique : calculation of the width of the managed area for a rectangular structure

This formula comes from the choice that is made to consider the upper integer number of passes required to manage the area. This configuration implies that the distance between two passes is always $W_{ch} + d_{floe}^{target}$ and that the real trajectory may be wider than the required management area[24]:

$$W_{traj}^{area} = N_{pass} \cdot W_{ch} + (N_{pass} - 1) \cdot d_{floe}^{target} \quad > \quad W_{traj}$$

Finally, please notice in Fig. 37b that the management trajectory differs if $N_{pass}$ has an odd or an even value.




(a) Calculation of $N_{pass}$

(b) Trajectory according to $N_{pass}$

Figure 37: Linear technique : trajectory

[24]Another solution would have been to reduce the resulting floe size until the trajectory width exactly matches with the area width, but 1) this requires tighter turns, and therefore more power, 2) the reduced floe size is not of any interest, while a wider trajectory enables more safety regards to a sudden change in the drift direction.

### *3.2.2 Length of the Managed Area*

The next step consists in the calculation of the length of the managed area $L_{area}$. For a given power level (and then the corresponding velocities in level ice), the objective is to determine, what is the required value for $L_{area}$ so that, after one management cycle, the ice has drift of exactly the same distance $L_{area}$. In other words, we are looking for the value of $L_{area}$ so that, when the vessel comes back to its initial position, it is in the appropriate position to start a new management cycle.

**Calculation** It should be noticed that, during a management cycle, one pass over two is performed updrift the ice motion, while the other half is performed downdrift, which results in different ice resistances, and then in different ship's velocities for a given power level. Then, $L_{area}$ can be expressed by

$$L_{area} = V_{drift} \cdot t_{cycle} = V_{drift} \cdot \left( \frac{\frac{1}{2} N_{pass} \left( L_{area} - 2R_{turning} \right)}{V_{lvl}^{updrift}} + \frac{\frac{1}{2} N_{pass} \left( L_{area} - 2R_{turning} \right)}{V_{lvl}^{downdrift}} + \cdots \right.$$
$$\left. \frac{W_{traj} - 2R_{turning}}{V_{lvl}^{nodrift}} + \frac{N_{pass} \pi R_{turning}}{V_{turning}} \right)$$

where

- $V_{lvl}^{updrift}$, $V_{lvl}^{downdrift}$ and $V_{lvl}^{nodrift}$ are respectively the ship's velocities when sailing in linear motion updrift, downdrift and without[25] relative ice motion. These velocities are computed from the *Equil_level_ice* function with a velocity respectively equal to $V_{ship} + V_{drift}$, $V_{ship} - V_{drift}$ and $V_{ship}$.
- $V_{turning}$ is the velocity of the ship in turning motion (cf. the two turning modes described in the manoeuvrability section, II.2.5.).

In the previous equation, $L_{area}$ appears on both sides of the equality. Solving for this value, we get

$$L_{area} = \frac{V_{drift} \cdot \left( \frac{-N_{pass} R_{turning}}{V_{lvl}^{updrift}} + \frac{-N_{pass} R_{turning}}{V_{lvl}^{downdrift}} + \frac{W_{traj} - 2R_{turning}}{V_{lvl}^{nodrift}} + \frac{N_{pass} \pi R_{turning}}{V_{turning}} \right)}{1 - V_{drift} \cdot \left( \frac{N_{pass}}{2V_{lvl}^{updrift}} + \frac{N_{pass}}{2V_{lvl}^{downdrift}} \right)}$$

**Selection** Fig. 38 shows an example of the influence of the delivered power and the drift speed on the required length of the managed area. As expected, $L_{area}$ depends strongly on the power level, and therefore on the achievable velocities in linear and turning motion. The main interrogation is now : how to select a convenient value for $L_{area}$ ? The following elements should be taken into consideration :

- A large value of $L_{area}$ is profitable, as it is related to a lower power level and increases the cycle time, and therefore reduces the frequency of manoeuvres (turns) to be carried out.

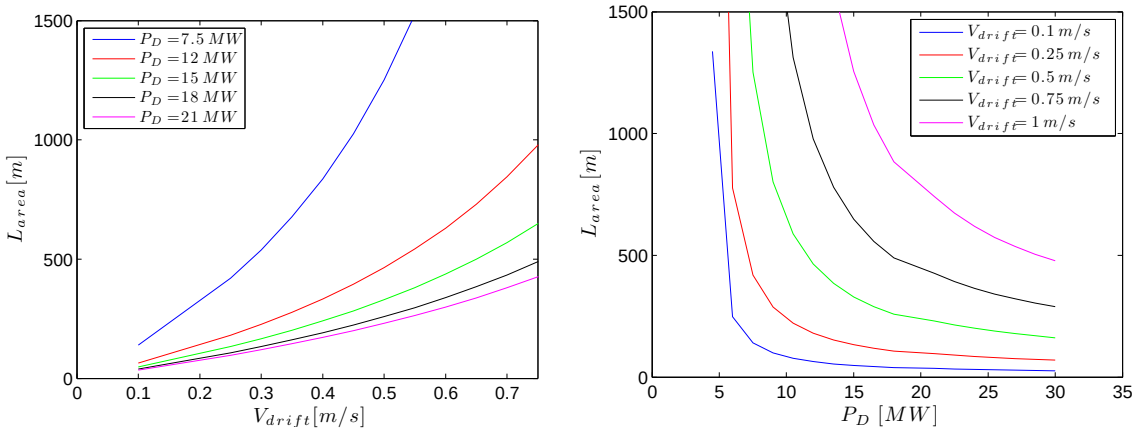
[25] Or, more accurately, when the drift direction is perpendicular to the the ship's motion.

– On the other hand, a large value of $L_{area}$ makes the ice management operations very sensitive to change in drift direction. Additionally, when the power level is too slow, a very tiny change in the delivered power induces a large change in the required value of $L_{area}$ (Fig. 38b).

Considering these elements, it has been decided to select the lowest possible value for $P_D$ so that a 10% error on its value does not induce more than 25% error on $L_{area}$ :

$$\begin{aligned} P_D &\rightarrow L_{area} \\ 90\%\, P_D &\rightarrow 125\%\, L_{area} \end{aligned}$$

This selection method can be considered as a *security* allowance on the delivered power. In the MatLab code, a specific function $L\_area\_fct$ has been written (cf. Appendix D) to estimate this value of $L_{area}$. It involves an iterative process to determine the power level that satisfies to the above condition.




(a) $L_{area}$ *vs* $V_{drift}$ (b) $L_{area}$ *vs* $P_D$

Figure 38: Influence of the delivered power and drift velocity on the length of managed area

**Minimum manageable floe size** Depending on the icebreaking vessel, the drift speed or the target floe size, the computed delivered power from $L\_area\_fct$ may be above the maximum available power of the vessel. In such a case, the software is able to determine what is actually the minimum manageable floe size. For this purpose, we progressively decrease the number of passes, compute the new floe size with

$$d_{floe}^{min,\,k} = \frac{W_{traj}}{N_{pass}^{k} - 1} - W_{ch} \qquad (iteration\ k)$$

and then recompute the length of managed area. This operation is repeated until a sufficiently low required power level is obtained (Fig. 39).

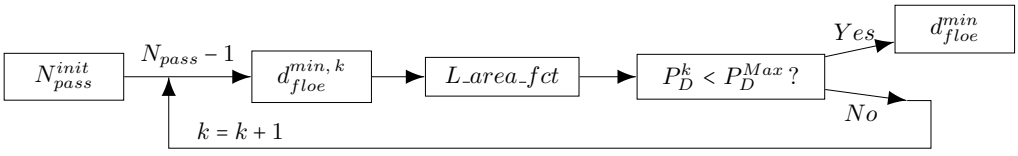



Figure 39: Linear technique : calculation of the minimum manageable floe size

**Maximum manageable area** When the requested icebreaking power is above the ship's maximum delivered power, another view of the problem is to determine which proportion of the requested managed area can be covered by the ice management vessel (close to full power), keeping the initial target floe size.

The algorithm is the following : as previously, the number of passes is decreased, but this time the new managed area width is recalculated at each iteration with

$$W_{area}^{max,\,k} = (N_{pass}^{k} - 1) \cdot (d_{floe}^{target} + W_{ch}) + W_{ch} \qquad (iteration\ k)$$

The length of the managed area is then computed and this operation is repeated until a sufficiently low required power level is obtained (Fig. 40). The resulting value of $W_{area}^{max}$ is very useful to determine which proportion of the requested area can be managed by the icebreaker, and therefore to evaluate if a second vessel is requested.

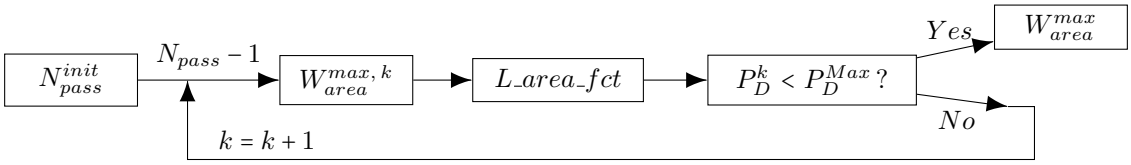



Figure 40: Linear technique : calculation of the maximum manageable area width

### *3.2.3 Simulation*

From all the above calculations, the simulation can now be performed for the linear ice management technique. The code related to this simulation is written in the *Ice_management_linear* and *Simulation_calc* functions. As these functions do not contain any specific code that would require description in this report, any interested reader is invited to directly have a look in the MatLab code of given in Appendix D.

**Calculation results** First, the results of the calculations are given in terms of power levels, velocities, turning radius and pods/rudder angle. An example of such results is given Fig. 41.

```
The optimal managed area length is 580.22 m
- Corresponding delivered power :
 * Linear motion: 9.84 MW
 * Turning:       9.84 MW (same as linear motion)
- Level ice velocities :
 * No relative drift : 4.89 m/s (9.51 kts)
 * Updrift :          4.39 m/s (8.54 kts)
 * Downdrift :        5.39 m/s (10.5 kts)
 * Turning :          1.24 m/s (2.41 kts)
- Turning radius : 118 m
- Pods angle : +/- 33.9 deg
```

Figure 41: Linear technique : example of calculation results

In case of insufficient icebreaking power, the software produces another output, with the maximum manageable area and possible solutions to perform the requested ice management, as shown in the example of Fig. 42.

```
SIMULATION ABORTED : Insufficient Icebreaking power. Unable to manage the ice.
With the given vessel and the requested target floe size, it is only possible
to manage 448m of the requested width of 698m (64.2%).
Possible solutions :
- increase icebreaking power up to at least 11.2 MW
- use more than one management vessel
- increase target floe size above 325 m.
```

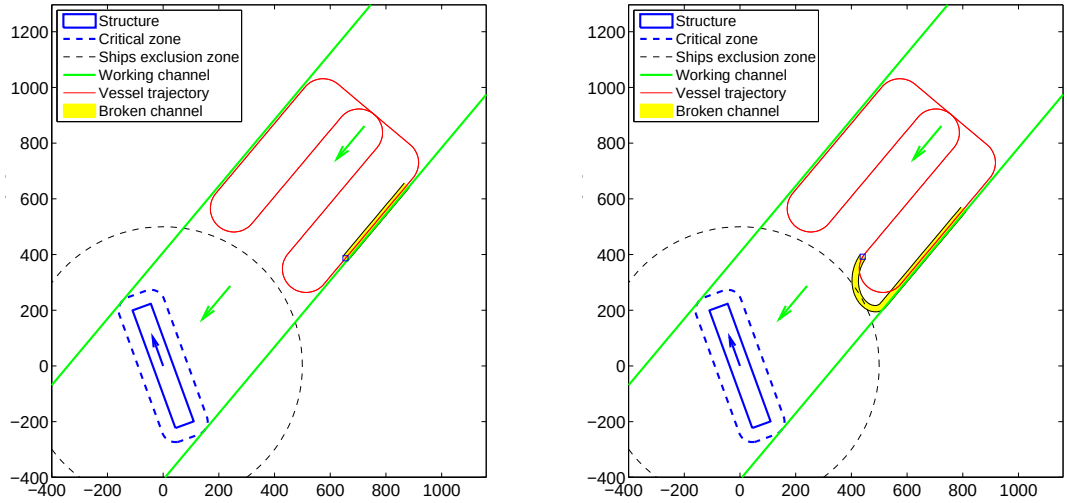
Figure 42: Linear technique : example of calculation results in case of exceeding requested power

**Ice drift and animation** After definition of the vessel trajectory, taking into account the velocities for updrift, downdrift, no drift and turning motion, it becomes then possible to generate an animation of the ice management operations.

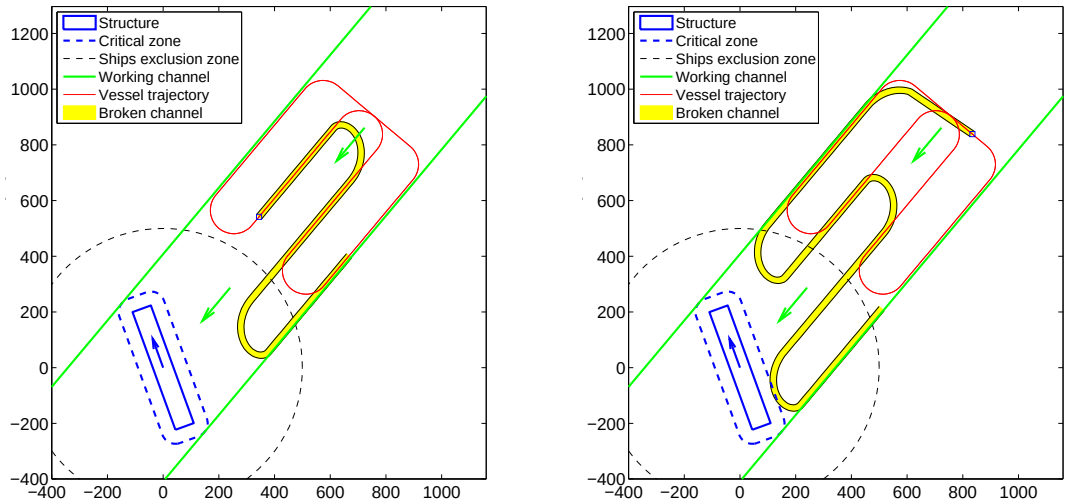
Basically, the simulation time is divided into successive time steps $dt$. For each time step, the new vessel position and its related broken channel are calculated. As the ice is drifting, each point $P = (P_x, P_y)$ of the broken channel is drifted after each time step by a small increment :

$$P^{(t)} = \begin{bmatrix} P_x^{(t)} \\ P_y^{(t)} \end{bmatrix} \quad \Rightarrow \quad P^{(t+1)} = \begin{bmatrix} P_x^{(t)} + V_{drift}\, dt\, \sin\alpha_{drift} \\ P_y^{(t)} + V_{drift}\, dt\, \cos\alpha_{drift} \end{bmatrix}$$

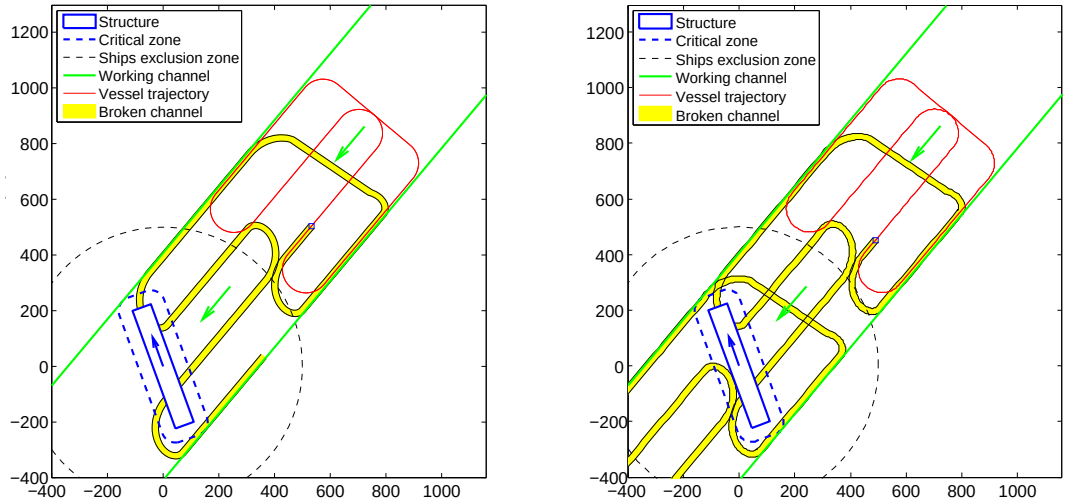
After calculation, an animation of the ice management operations is obtained. An example of such an animation is given in Fig. 43.




(a) $t = 1\,min$ (b) $t = 5\,min$




(c) $t = 12\,min$ (d) $t = 20\,min$




(e) $t = 27\,min$ (f) $t = 50\,min$

Figure 43: Linear Ice Management Technique : visualization

## 3.3 Double Pass Linear Technique

One of the main disadvantages of the linear technique described in the previous section is that it often requires very tight turns of the icebreaking vessel in order to achieve sufficient management of the drifting ice. The turning radius was actually calculated as

$$R_{turning}^{single\,pass} = \frac{d_{floe}^{target} + W_{ch}}{2}$$

so that, when the target floe size is small, the turning radius may become too small for the icebreaker, so that it is either impossible to achieve the ice management, or the turns simply requires an excessive power.

### *3.3.1 Presentation*

In order to enable ice management with small target floe sizes, a new approach is suggested in Fig. 44, called the *double pass linear management technique*. It consists, after a first management cycle, in passing through the middle of the already broken floes to further reduce their size. The new turning radius is then

$$R_{turning}^{double\,pass} = d_{floe}^{target} + W_{ch} = 2\,R_{turning}^{single\,pass}$$

The result is, for constant floe size, a lower required delivered power in turns AND in linear motions[26], or for constant power, a smaller achievable floe size.

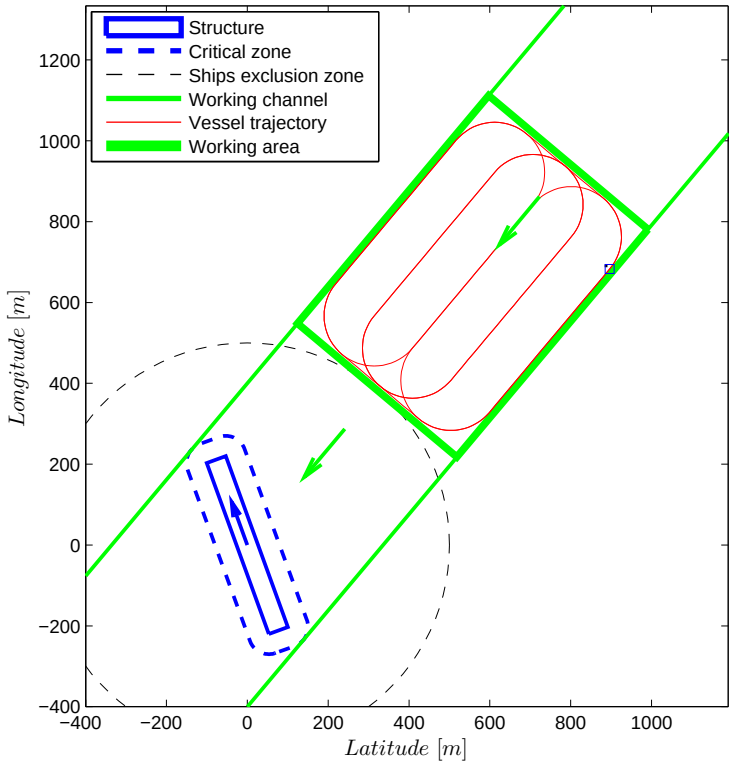



Figure 44: Double Pass Linear Management Technique

[26] When the icebreaking vessel is advancing in already broken floes, the ice resistance is lower, so that the mean power level during a management cycle is decreased.

### *3.3.2 Main Modifications*

As the double pass linear technique is quite similar to the single pass linear technique, focus is made here only on the main changes that have been carried out to this previous technique.

**Mean velocity** When observing an animation of the double pass technique (cf. below), it is observed that approximately 50% of the trajectory of the icebreaking vessel is sailed in floes instead of level ice. For this reason, we keep the same formulae as for the single pass technique, but considering new mean velocities for linear and turning motions :

$$V_{mean}^{updrift} = \frac{2}{\frac{1}{V_{lvl}^{nodrift}} + \frac{1}{V_{floes}^{nodrift}}} \qquad V_{mean}^{downdrift} = \dots$$

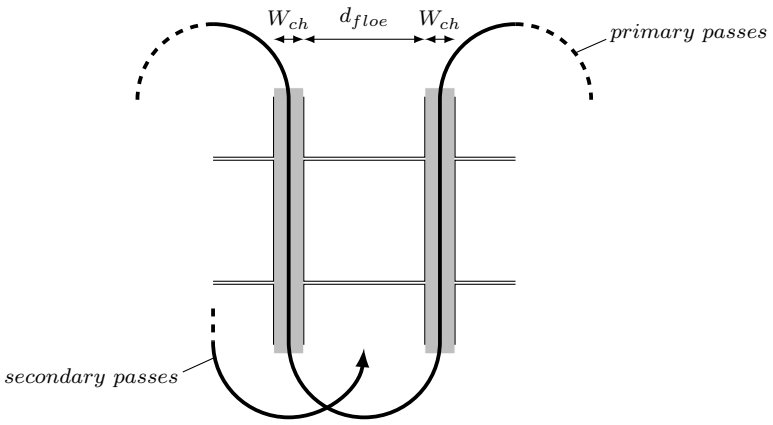
These formulas are actually the harmonic means of the velocities in level ice $V_{lvl}^{\dots}$ and in floes $V_{floes}^{\dots}$.

**Velocities in floes** The velocities in floes are calculated in the *Equil_floes* function, with the same method as for the equilibrium velocities in level ice (*Equil_level_ice*, cf. II.2.2.10), but considering this time the resistance in floes. This resistance is estimated with the method detailed in section II.2.3 :

$$R_{ice} = Ice\_resistance\_floes\,(V,\ d_{floe},\ C_{ice},\ Ship\ geometry,\ Ice\ properties)$$

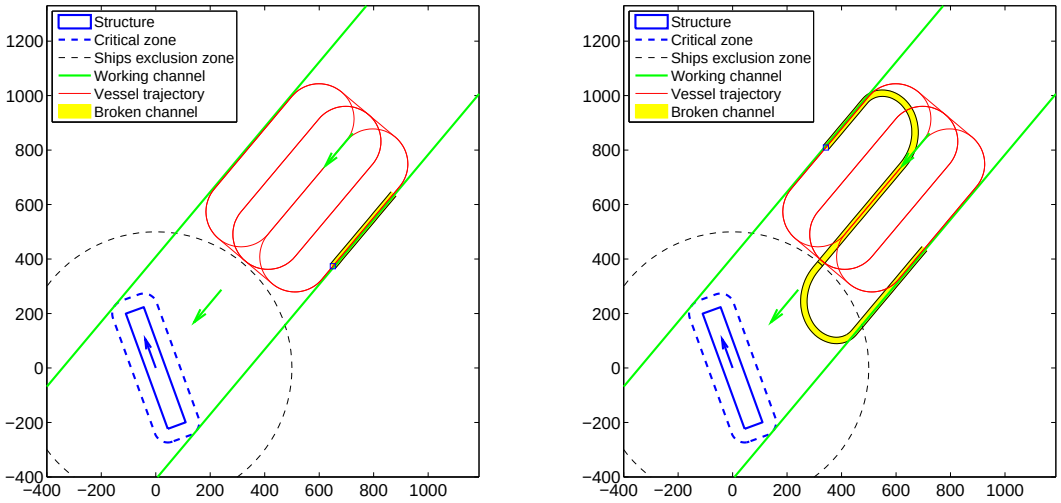
with the following parameters :

- $d_{floe} = 2\,d_{floe}^{target}$, the resulting floe size after the primary passes
- $C_{ice} = \dfrac{d_{floe}^2}{d_{floe}^2 + W_{ch}\,d_{floe}}$, the estimated ice concentration, considering the width $W_{ch}$ of the previously broken channel :

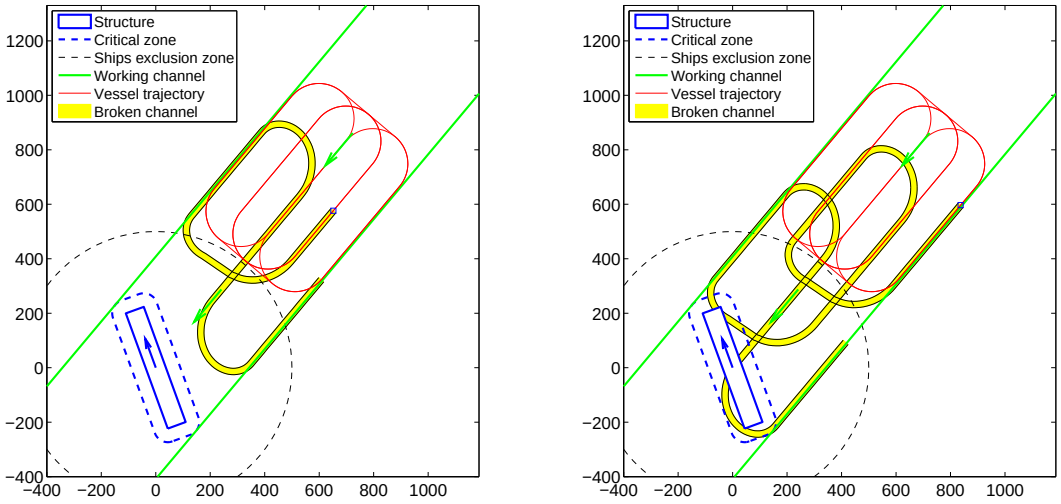



### *3.3.3 Simulation*

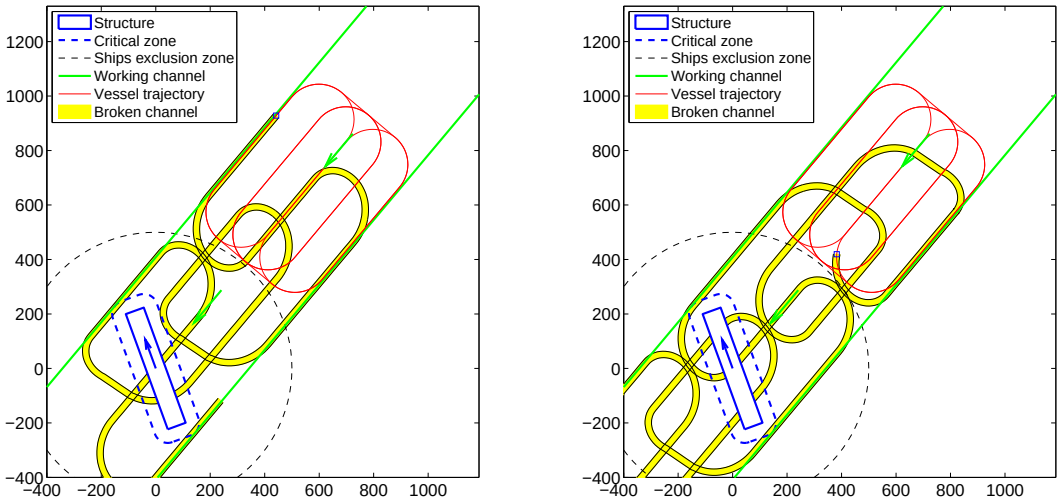
The simulation is then calculated and an animation of the ice management operations is obtained. An example of such an animation is given in Fig. 45.




(a) $t = 1\,min$ (b) $t = 10\,min$




(c) $t = 15\,min$ (d) $t = 25\,min$




(e) $t = 35\,min$ (f) $t = 50\,min$

Figure 45: Double Pass Linear Ice Management Technique : visualization

## 3.4 Sector Technique

The next technique that has been implemented in the software is the sector technique. In this case, the vessel breaks the ice perpendicularly to the drift direction, generating a wide managed ice area in front of the structure to defend. This technique is to be preferred when ice drift speed is low and/or drift direction is unstable, as the cycle time is relatively short and it is therefore easy to modify the shape and the orientation of the trajectory.

As it can be observed on Fig. 46, the trajectory required to obtain a broken channel mainly perpendicular to the drift direction has the shape of a 8-loop.

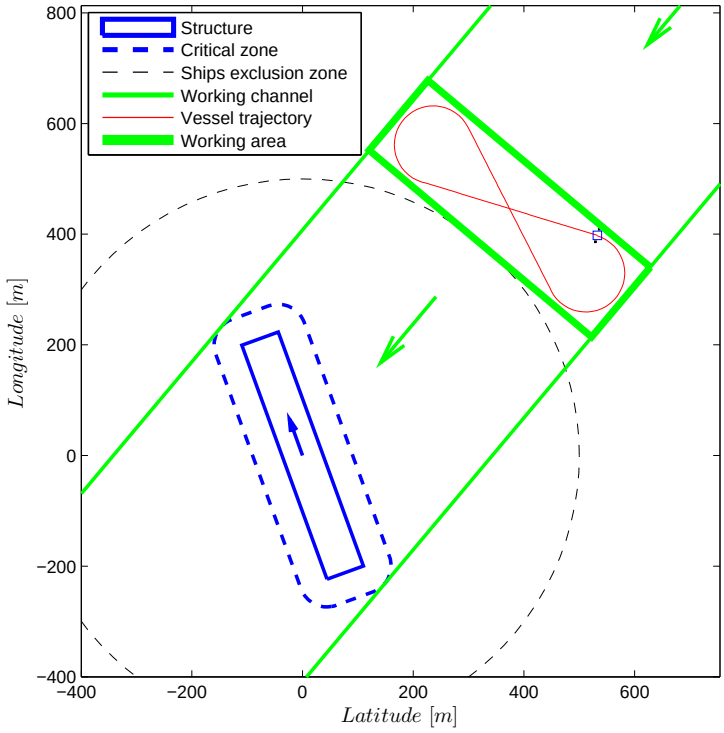



Figure 46: Sector management technique

### *3.4.1 Area To Be Managed*

As the sector technique is mainly used in case of a varying drift direction, a new parameter must be introduced, the maximum rate of change of the drift direction angle :

$$\dot{\alpha}_{drift}^{max}$$

When the drift direction is varying, the width of the management area must be enlarged so that, if a sudden change in the direction occurs, the structure will still be protected again the incoming level ice. From the diagram of Fig. 47, we deduce that[27,28] :

$$W_{area} = \begin{cases} W_{area}^{lin} + 2\dfrac{R_{excl} + R_{turning}}{\tan\theta} & \textit{if circular structure} \\ W_{area}^{lin} + |DD'| + |FF'| & \textit{if rectangular structure} \end{cases}$$

[27]The formula for a circular structure can be easily deduced by some simplifications of Fig. 47.
[28]Note that the formula is different for some specific orientations of the structure (see code).

with $W_{area}^{lin}$ the managed area width calculated for the linear technique, and

$$|DD'| = \frac{|AD|}{\tan\theta_1} = \frac{R_{excl} + R_{turning} - |OC|}{\tan\theta_1}$$
$$|FF'| = \frac{|EF|}{\tan\theta_2} = \frac{R_{excl} + R_{turning} + |OB|}{\tan\theta_2}$$

where

$$|OB| = |OC| = |OA|\cos(\alpha_1 + \alpha_2) = \sqrt{(B/2 + t_{cr})^2 + (L/2 + t_{cr})^2}\cos(\alpha_1 + \alpha_2)$$
$$\alpha_1 = \alpha_{drift} - \pi - \psi^{struct} \quad and \quad \alpha_2 = \tan^{-1}\frac{\frac{B}{2} + t_{cr}}{\frac{L}{2} + t_{cr}}$$
$$\theta_1 = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{|AD|}{V_{drift}}\ , \quad \theta_2 = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{|EF|}{V_{drift}} \quad and \quad \theta = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{R_{excl} + R_{turning}}{V_{drift}}$$

The width of the trajectory is finally

$$W_{traj} = W_{area} - W_{ch}$$

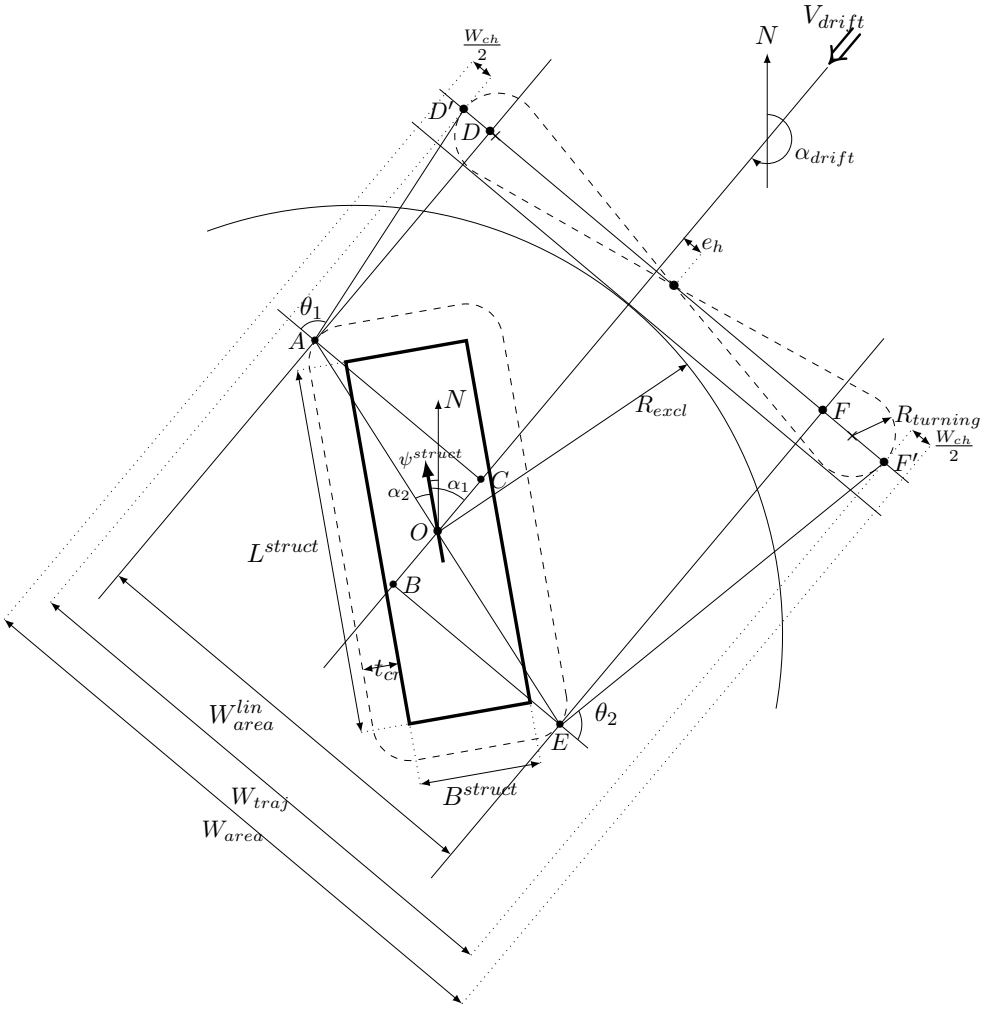



Figure 47: Sector technique : calculation of the width of the managed area

The angles $\theta_1$, $\theta_2$ (for rectangular structure) and $\theta$ (for circular structure) are calculated so that, if the drift direction starts changing at a rate of $\dot{\alpha}_{drift}^{max}$, the ice broken in point $D'$ or $F'$ will reach respectively point $A$ or $E$, and the structure will therefore still be protected against level ice. As the variation is assumed to be done progressively, the angles $\theta_1$, $\theta_2$ and $\theta$ are calculated by $\frac{\pi}{2}$ minus the area (integral) of the triangle represented in Fig. 48.

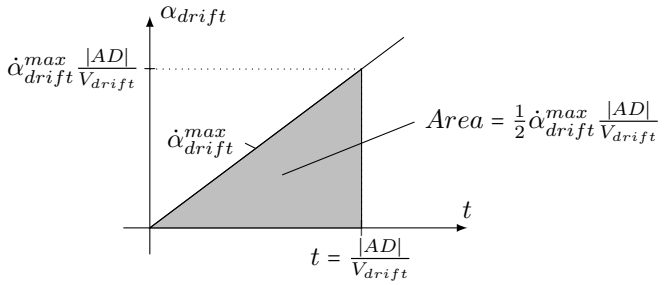



Figure 48: Variation of the drift direction (example for $\theta_1$)

Additionally, for rectangular structures, as one corner is generally further away from the trajectory, there an offset $e_h$ between the centre of the trajectory and the centre of the structure, so that the trajectory is shifted on the side of the corner that is more away (and therefore that must be more protected, Fig. 47). This offset is calculated with

$$e_h = \frac{|FF'| - |DD'|}{2} = \frac{1}{2}\left(\frac{R_{excl} + R_{turning} + |OB|}{\tan\theta_2} - \frac{R_{excl} + R_{turning} - |OC|}{\tan\theta_1}\right)$$

### *3.4.2 Trajectory*

As already shown previously, the trajectory has the shape of a 8-loop with a given width $W_{traj}$ and turning radius $R_{turning}$ (Fig. 49). This subsection is dedicated to the accurate calculation of this trajectory.

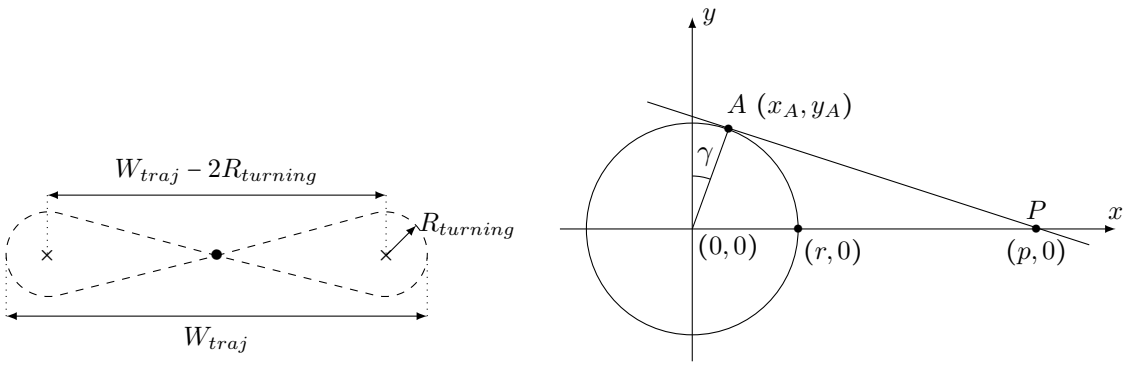



Figure 49: Sector technique : details of the trajectory

**Turning radius** The first element to consider is that, as the ice is drifting, the turning radius cannot be simply the half of $d_{floe}^{target} + W_{ch}$. Considering that the turns are carried out updrift

the ice motion[29], we have

$$\begin{aligned} R_{turning} &= \frac{1}{2}\left(d_{floe}^{target} + W_{ch}\right) - \frac{1}{2}V_{drift} \cdot time\ to\ turn\ 180^o \\ &= \frac{1}{2}\left(d_{floe}^{target} + W_{ch}\right) - \frac{1}{2}V_{drift}\frac{\pi R_{turning}}{V_{turning}} \end{aligned}$$

Note that, as the angle $\gamma$ (cf. Fig. 49b) is still unknown at this point, we make the simplification that the turns are only $180^o$ instead of $180^o + 2\gamma$. Solving for $R_{turning}$, we obtain

$$R_{turning} = \frac{\frac{1}{2}\left(d_{floe}^{target} + W_{ch}\right)}{1 + \frac{\pi}{2}\frac{V_{drift}}{V_{turning}}}$$

so that $R_{turning}$ logically decreases when the drift speed increases. It is also important to notice that the turning radius is no more independent of the turning speed.

**Turning arcs** Considering the conventions given in Fig. 49b, it is possible to determine the position of point $A$ and the associated angle $\gamma$. The equations of the circle and the straight line can be expressed by

$$Circle : x^2 + y^2 = r^2$$
$$Straight\ line : y = mx + k$$

The point $A$ is considered as the unique intersection of the straight line with the circle so that, after some developments, we find

$$k = -mp \qquad and \qquad m = \sqrt{\frac{r^2}{p^2 - r^2}}$$

and finally

$$x_A = \frac{r^2}{p} \qquad y_A = \frac{-r}{p}\sqrt{p^2 - r^2} \qquad \gamma = \tan^{-1}\frac{x_A}{y_A}$$

This result is then used in the MatLab code to draw the trajectory, with

$$r = R_{turning} \qquad and \qquad p = \frac{W_{traj} - 2R_{turning}}{2}$$

considering a coordinate system of origin at the trajectory center.

### *3.4.3 Power Level*

In the previous subsection, it can be noticed that, once the turning velocity is defined, it is possible to compute the turning radius and then the trajectory. However, the turning velocity is not independent on the trajectory, as explained in the following.

[29] When the trajectory is followed in the opposite way, with downdrift turns, the resulting broken field is not adequate.

Reminding that $d_{floe}^{target}$ is the target floe size, the trajectory must be so that

$$V_{drift}\,\frac{t_{cycle}}{2} = d_{floe}^{target} + W_{ch}$$

where $t_{cycle}$ is the cycle time. The above equation expresses the fact that during half of a cycle, the ice must have drifted by $d_{floe}^{target} + W_{ch}$. The velocity along the trajectory, and therefore the power level must then be selected so that

$$t_{cycle} = \frac{2(d_{floe}^{target} + W_{ch})}{V_{drift}}$$

On the other hand, the cycle time is calculated from the geometry of the trajectory :

$$t_{cycle} = \frac{4\sqrt{x_A^2 + y_A^2}}{V_{lvl}} + \frac{(2\pi + 4\gamma)R_{turning}}{V_{turning}}$$

In the above equation, it should be reminded that $x_A$, $y_A$ and $\gamma$ depend on $R_{turning}$, that $R_{turning}$ depends on $V_{turning}$, and that both $V_{turning}$ and $V_{lvl}$ depend on the power level, so that the problem is highly nonlinear. For this reason, in the MatLab code, a specific function *Calc_P_D_sector* is used to evaluate

$$\frac{4\sqrt{x_A^2 + y_A^2}}{V_{lvl}} + \frac{(2\pi + 4\gamma)R_{turning}}{V_{turning}} - \frac{2(d_{floe}^{target} + W_{ch})}{V_{drift}}$$

as a function of $P_D$, and for a defined turning mode (cf. II.2.5). The function `fzero` is then used to determine the value of the power level inducing the above expression to be zero.

### *3.4.4 Minimum Manageable Floe Size*

If the required power level determined in the previous subsection is larger than the maximum power level of the ice management vessel, the software computes the minimum manageable floe size, considering the vessel working at maximum power. The objective is then to solve the following equation for $d_{floe}^{min}$ :

$$\frac{4\sqrt{x_A^2 + y_A^2}}{V_{lvl}^{P_D^{max}}} + \frac{(2\pi + 4\gamma)R_{turning}^{P_D^{max}}}{V_{turning}^{P_D^{max}}} = \frac{2(d_{floe}^{min} + W_{ch})}{V_{drift}}$$

with $V_{lvl}^{P_D^{max}}$ and $V_{turning}^{P_D^{max}}$ the achievable velocities at maximum power respectively in linear and in turning motions. $R_{turning}^{P_D^{max}}$ is then the lowest achievable value of the trajectory radius at maximum delivered power level. It should also be noted that

- $x_A$, $y_A$ and $\gamma$ are function of $R_{turning}^{P_D^{max}}$, and that
- $R_{turning}^{P_D^{max}} = \dfrac{\frac{1}{2}\left(d_{floe}^{min} + W_{ch}\right)}{1 + \frac{\pi}{2}\frac{V_{drift}}{V_{turning}^{P_D^{max}}}}$

In order to solve this complex problem, a few simplifications are made. First, the angle $\gamma$ of the turning arc is neglected regards to $\pi$. Then, the linear parts of the trajectory (two lengths of $2\sqrt{x_A^2 + y_A^2}$) are approximated by a rough estimation of $W_{traj}$ :

$$W_{traj}^{estim} = W_{area}^{cst} + \frac{2R_{turning}^{P_D^{max}}}{\tan\theta_m}$$

with the constant component $W_{area}^{cst} = W_{area}^{lin} + 2\frac{R_{excl}}{\tan\theta_m}$ and the mean angle $\theta_m = \frac{1}{2}(\theta_1 + \theta_2)$. Finally, solving for $d_{floe}^{min}$, we get

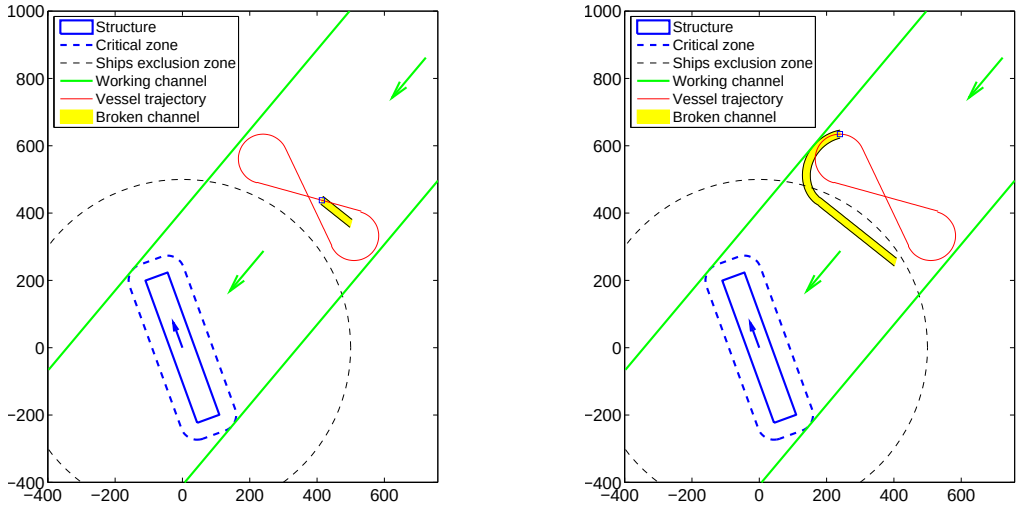
$$d_{floe}^{min} = -W_{ch} + \frac{W_{area}^{cst}}{V_{lvl}^{P_D^{max}} \left( \frac{1}{V_{drift}} - \frac{\frac{1}{\tan\theta_m V_{lvl}^{P_D^{max}}} + \frac{\pi}{2V_{turning}^{P_D^{max}}}}{1 + \frac{\pi}{2}\frac{V_{drift}}{V_{turning}^{P_D^{max}}}} \right)}$$

### *3.4.5 Simulation*

The simulation is then calculated in the MatLab script *Ice_management_sector* and an animation of the ice management operations according to the sector technique is obtained. An example of such an animation is given in Fig. 50, without any variation of the drift direction.

An example with variable drift direction is given Fig. 51. In this case, the drift angle varies from $220^o$ to $265^o$. The trajectory is automatically rotated to face the drift direction[30]. From a steady-state configuration in Fig. 51a, the drift direction starts changing (Fig. 51b), but the drifting direction of the already broken floes will also change, instead of following their initial direction. As the width of the trajectory has been calculated to be larger than $W_{area}^{lin}$, we observe in Fig. 51c, 51d and 51e that the structure is still protected against level ice. Finally, if the drift direction stabilizes (Fig. 51f), we come back to a configuration close to the initial one.

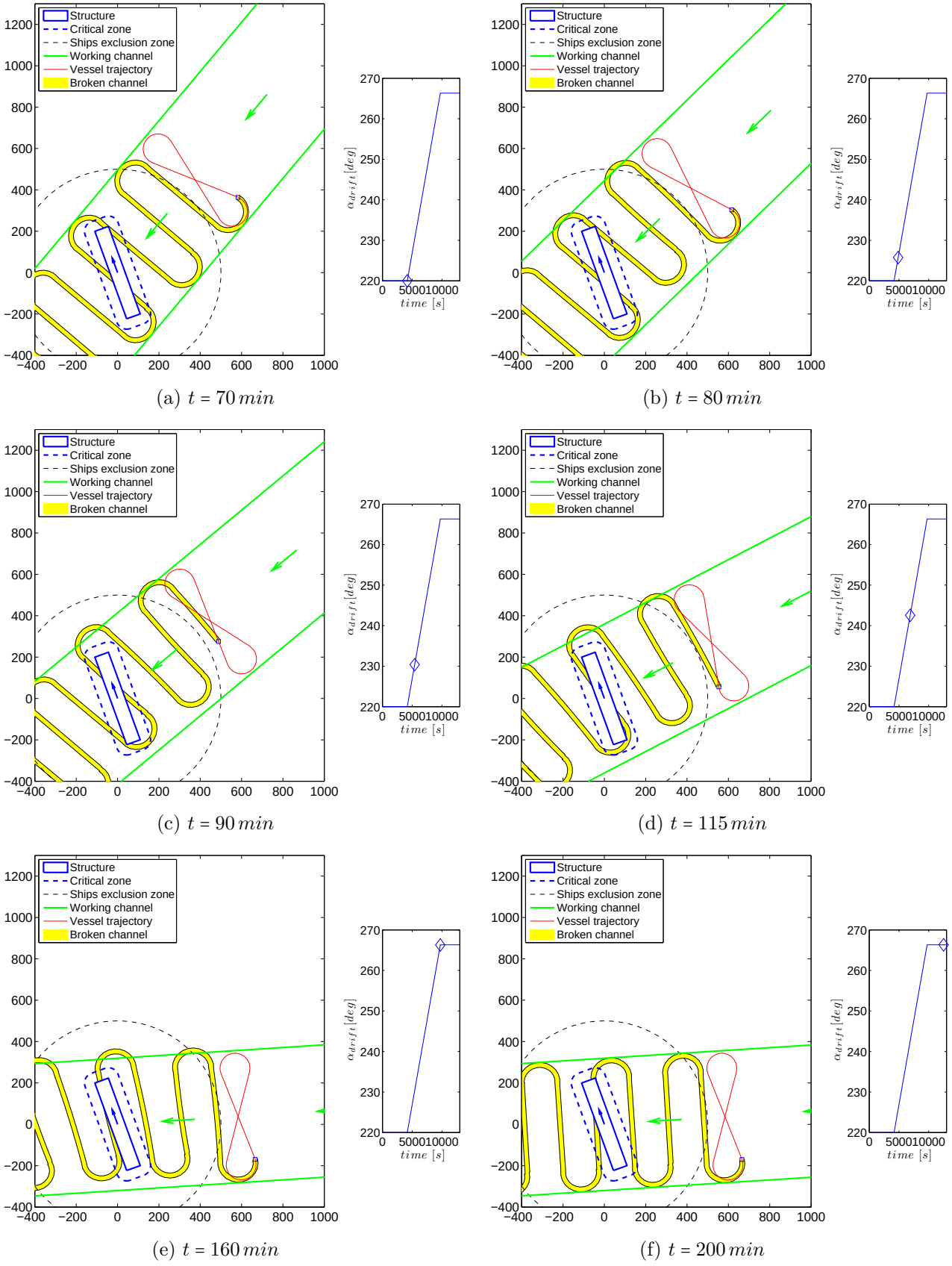
[30]For simplification purposes, it is assumed that the ice management vessel receives informations about the instantaneous drift direction and is able to simultaneously modify its trajectory.




(a) $t = 1\,min$ (b) $t = 5\,min$

(c) $t = 12\,min$ (d) $t = 40\,min$

Figure 50: Sector Ice Management Technique : visualization




(a) $t = 70\,min$ (b) $t = 80\,min$

(c) $t = 90\,min$ (d) $t = 115\,min$

(e) $t = 160\,min$ (f) $t = 200\,min$

Figure 51: Sector Ice Management Technique : visualization (variable drift direction)

## 3.5 Circular Technique

The last technique implemented for the present Master's thesis is the circular technique. With this technique, the vessel moves in circular patterns updrift the platform location (Fig. 52). This technique is mainly used in case of high concentration of thin ice or small but thick ice floes when drift direction is variable.

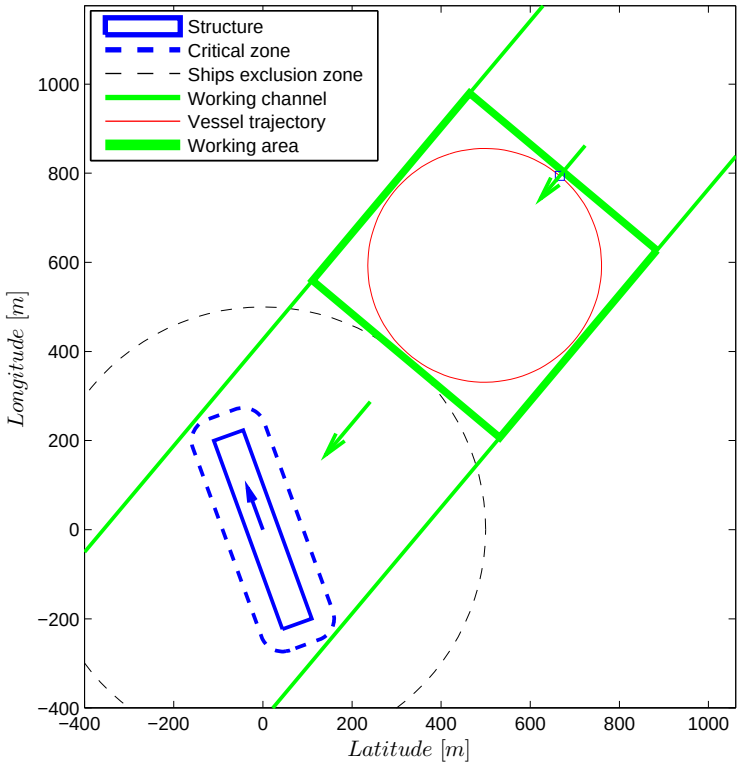



Figure 52: Circular management technique

### *3.5.1 Trajectory Radius*

With the circular technique, the trajectory is only defined by its radius $R_{traj}$. According to the drawing of Fig. 53, the width of the trajectory is simply

$$W_{traj} = 2R_{traj}$$

but also[31]

$$W_{traj} = \begin{cases} W_{area}^{lin} + 2\dfrac{R_{excl} + D_{struct}/2 + t_{cr} + R_{traj}}{\tan\theta} & \textit{if circular structure} \\ W_{traj}^{lin} + \dfrac{R_{excl} - |OC| + R_{traj}}{\tan\theta_1} + \dfrac{R_{excl} + |OB| + R_{traj}}{\tan\theta_2} & \textit{if rectangular structure} \end{cases}$$

[31] Again, the formula for a circular structure can be easily deduced by some simplifications of Fig. 53, and please also note that the formula is different for some specific orientations of the structure (see code).

with $W_{area}^{lin}$ the managed area width calculated for the linear technique, and

$$|OB| = |OC| = |OA|\cos(\alpha_1 + \alpha_2) = \sqrt{(B/2 + t_{cr})^2 + (L/2 + t_{cr})^2}\cos(\alpha_1 + \alpha_2)$$

$$\alpha_1 = \alpha_{drift} - \pi - \psi^{struct} \quad and \quad \alpha_2 = \tan^{-1}\frac{\frac{B}{2} + t_{cr}}{\frac{L}{2} + t_{cr}}$$

$$\theta_1 = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{R_{excl} - |OC| + R_{traj}}{V_{drift}} \quad and \quad \theta_2 = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{R_{excl} + |OB| + R_{traj}}{V_{drift}}$$

$$\theta = \frac{\pi}{2} - \frac{1}{2}\dot{\alpha}_{drift}^{max}\frac{R_{excl} + D^{struct} + t_{cr} + R_{traj}}{V_{drift}}$$

For rectangular structures, the offset $e_h$ between the centre of the trajectory and the centre of the structure is calculated with

$$e_h = \frac{|DD'| - |FF'|}{2} = \frac{R_{excl} + |OC| + R_{traj}}{\tan\theta_1} - \frac{R_{excl} - |OB| + R_{traj}}{\tan\theta_2}$$

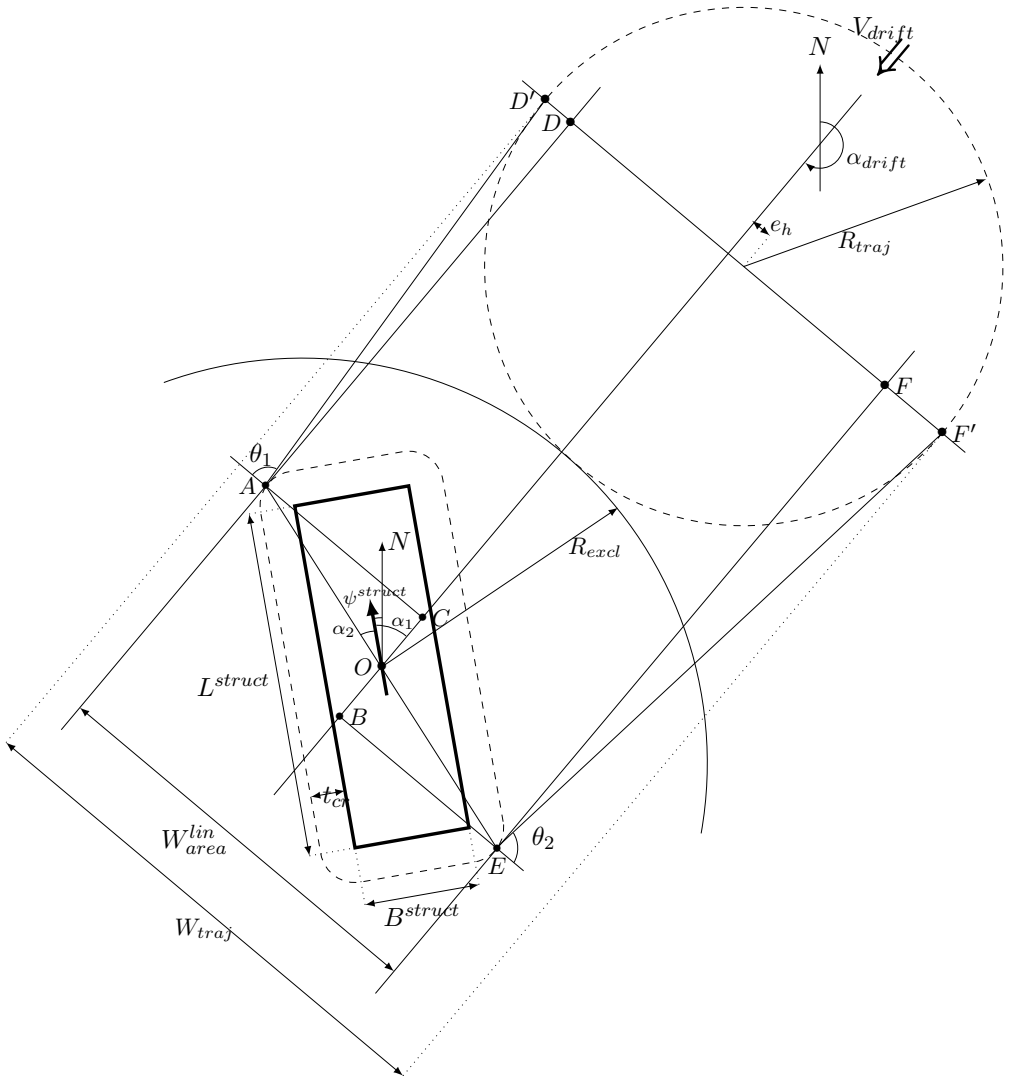



Figure 53: Circular technique : calculation of the trajectory radius

It is observed that the trajectory radius appears in both expressions of $W_{traj}$. For this reason, the `fzero` MatLab function is used to solve the problem and compute the appropriated value of $R_{traj}$.

### *3.5.2 Ship's Velocity*

The next step consists in the calculation of the ship's velocity along the circular trajectory. Compared to the previous ice management techniques, the calculation is here relatively more complex, as the ship's velocity cannot be anymore calculated so that

$$V_{drift} \cdot t_{cycle} = d_{floe}^{target} + W_{ch}$$

For the circular ice management technique, we have indeed to consider that, according to the ship's velocity, the largest floes may appears in three different positions on the resulting broken ice pattern, as shown in Fig. 54. From this figure, the size of each of the three potential largest floes can be estimated as

$$Floe\ 1 : d_1 = |AB| = |OG| - |CG| - W_{ch} = \mathrm{mod}(|OG|, |OF|) - W_{ch}$$

$$Floe\ 2 : d_2 = |DE| = |OF| - |OC| - W_{ch} = |OF| - \mathrm{mod}(|OG|, |OF|) - W_{ch}$$

$$Floe\ 3 : d_3 = \frac{|A'E'|}{\sqrt{2}} = \frac{|OF| - W_{ch}}{\sqrt{2}}$$

with

$$|OG| = 2R_{traj} - V_{drift}\frac{\pi R_{traj}}{V_{traj}} \quad and \quad |OF| = V_{drift}\frac{2\pi R_{traj}}{V_{traj}}$$

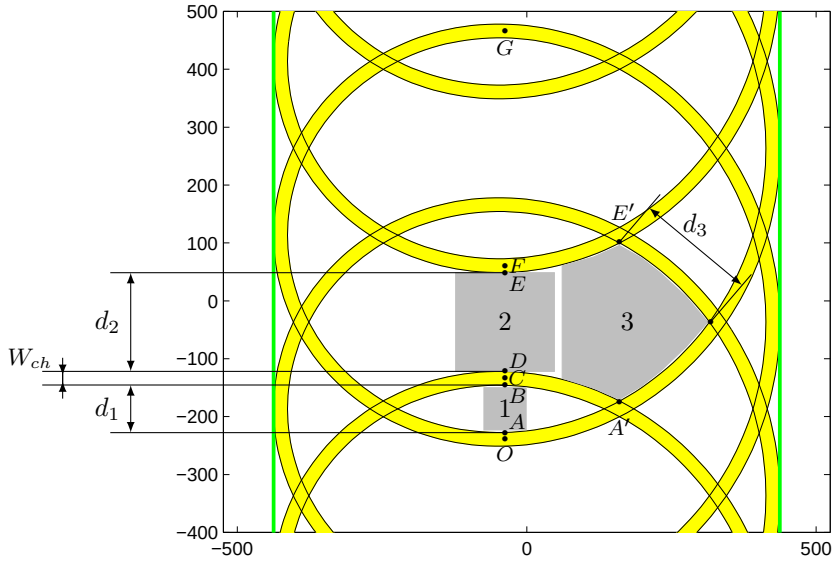



Figure 54: Circular technique : the three possible positions for the largest generated floes

As $|OG|$ and $|OF|$ depend on $V_{traj}$, the size of these three floes will also depend on $V_{traj}$, for a given trajectory radius. An example of the floes sizes evolution is given in Fig. 55. The ship's velocity that has to be selected for the ice management operations is then the largest value of

$V_{traj}$ that satisfies

$$\max(d_1, d_2, d_3) \leq d_{floe}^{target}$$

In the example of Fig. 55, this velocity is $4.8\,knots$.

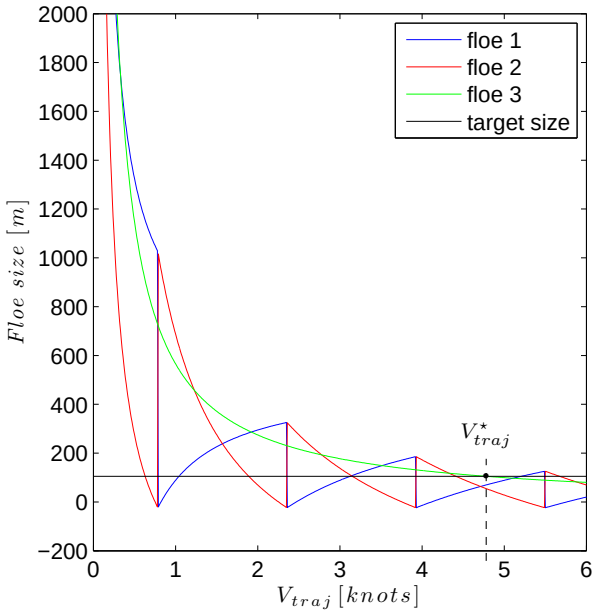



Figure 55: Circular technique : influence of the turning velocity on the resulting maximum floe size

### *3.5.3 Power Level*

Once the required ship's velocity is known, it is relatively easy to evaluate the required power level. Following the method described in the manoeuvrability section (cf. II.2.5), the requested power is obtained with

$$P_D = Calc\_circle\_V(R_{traj}, V_{traj})$$

**Minimum manageable floe size** In the case of a too large requested power compared to the ship's maximum power, the software computes the minimum manageable floe size. For this purpose, the software simply considers the smallest manageable floe size that is achievable with $V \leq V^{P_D^{max}}$, with $V^{P_D^{max}}$ the achievable velocity at maximum power level. Note that the minimum manageable floe size is not necessarily obtained for $V^{P_D^{max}}$ (cf. Fig. 55, curves related to floes 1 and 2 are not monotonically increasing for decreasing velocity.).

**Maximum manageable area** As previously, the software determines also the maximum width that can be managed with the requested floe size, as this is a crucial information to decide whether or not a second ice management vessel should be used. This is performed by taking the largest value of the trajectory radius that requires a power level equal to the ship's maximum power, as shown in the example[32] of Fig. 56.

[32]The piecewise behavior of the curve observed in Fig. 56 is mainly the result of the nonlinearity of the floe size versus ship's velocity (Fig. 55), but also of the simultaneous variation of trajectory radius and ship's velocity, acting in opposite direction on the requested power.

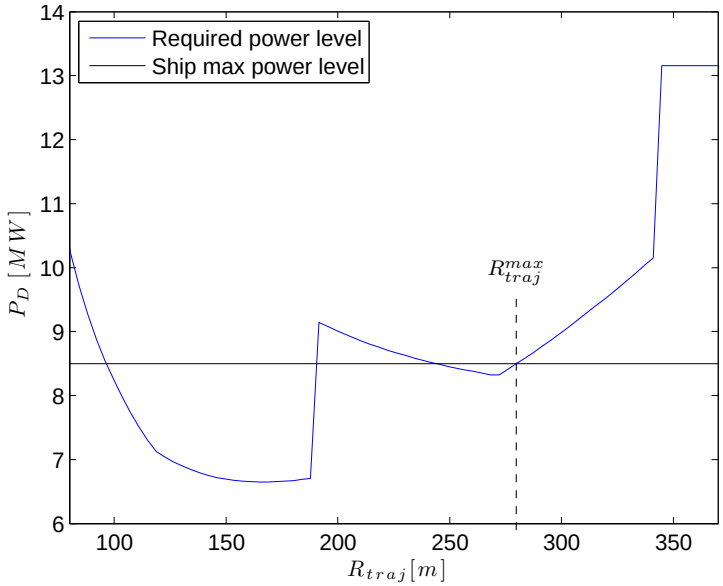
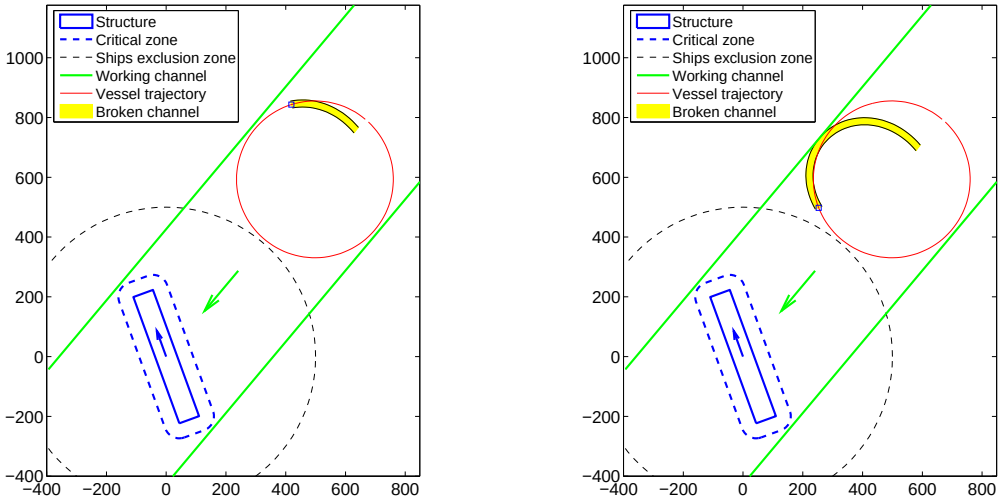



Figure 56: Circular technique : calculation of maximum manageable area

### *3.5.4 Simulation*

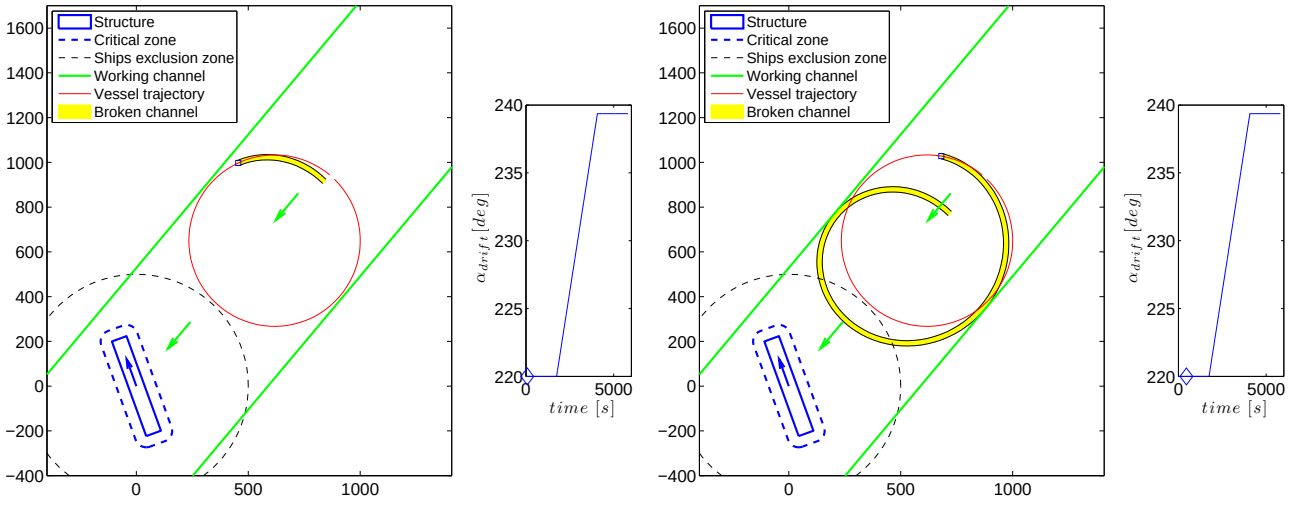
After calculation of the simulation in the MatLab script *Ice_management_circular*, an animation of the ice management operations according to the circular technique is obtained. Examples of such an animation are given in Fig. 57 for constant drift direction and in Fig. 58 with variation of the drift direction and instantaneous modification of the centre of the trajectory. Again, we observe that the structure is still protected against the drifting level ice during the operations.



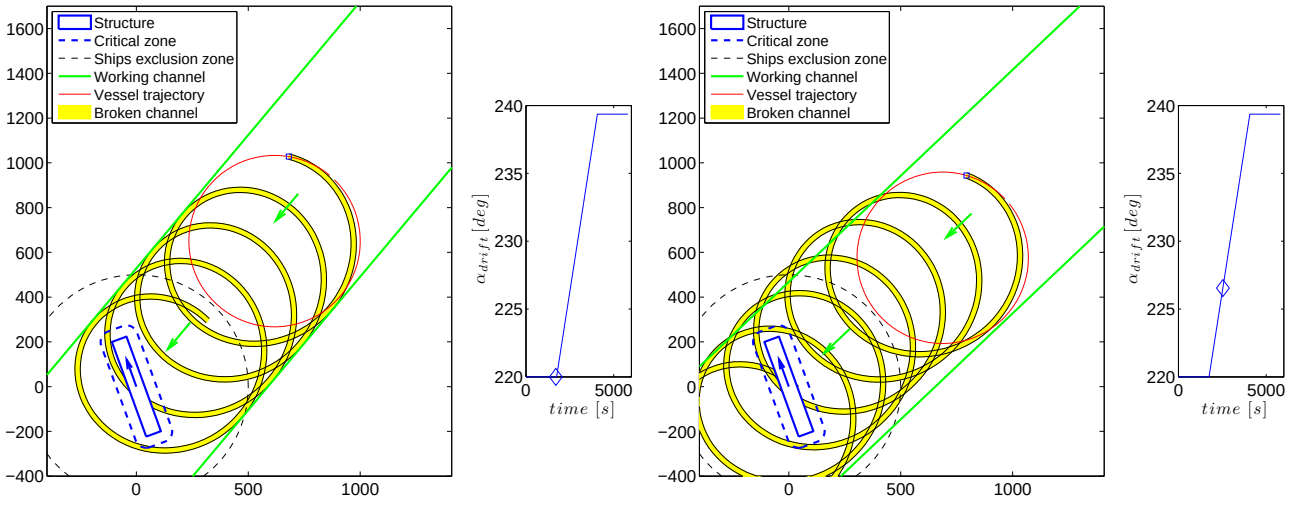

(a) $t = 1\,min$ (b) $t = 5\,min$

(c) $t = 10\,min$ (d) $t = 50\,min$

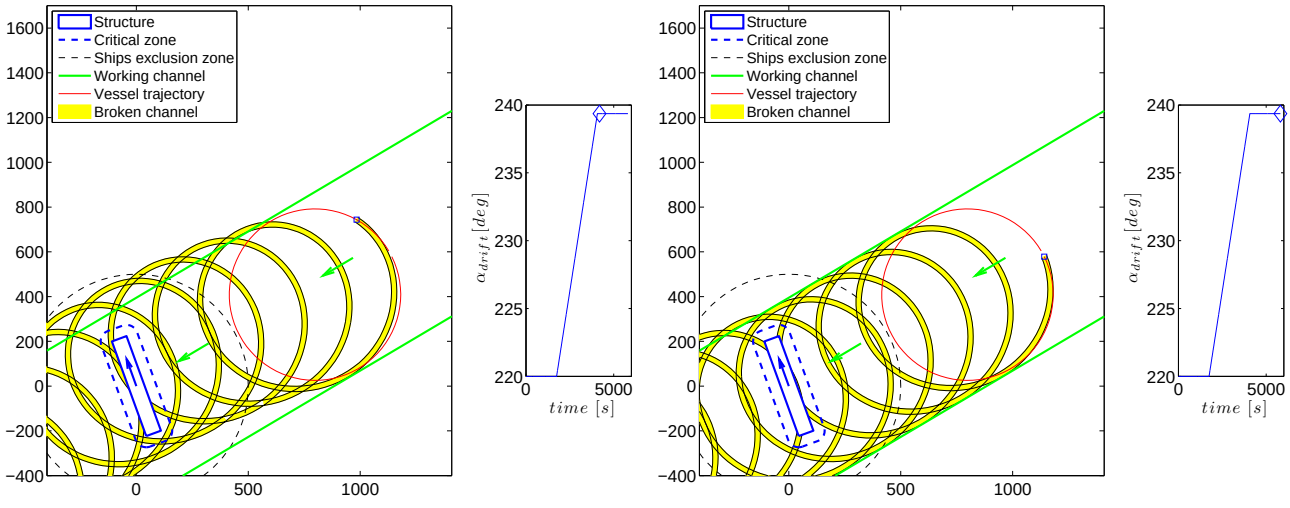
Figure 57: Circular Ice Management Technique : visualization




(a) $t = 1\,min$ (b) $t = 7\,min$




(c) $t = 30\,min$ (d) $t = 40\,min$




(e) $t = 70\,min$ (f) $t = 100\,min$

Figure 58: Circular Ice Management Technique : visualization (variable drift direction)

# 4 RESULTING FLOE SIZE ANALYSIS

After simulation of the ice management techniques, it is interesting to analyze the resulting floe size distribution. The objective is, from the calculated pattern of broken ice, to calculate the size, mass and momentum distributions of the resulting floes. Please remind that simulations are calculated so that level ice is broken in stripes, and these stripes are assumed to automatically break into more or less square floes (Fig. 34, section II.3.1.3).

## 4.1 Methodology

### *4.1.1 Discretization*

The first step consists in post-processing the results of the simulation. From the numerical values of the coordinates of the broken channel, a discretized matrix of the managed ice has to be generated. The process involves the following successive steps :

1. The inputs of the calculation are the outputs of the simulation, that is to say, the coordinates of the two sides of the broken channel (Fig. 59a).

2. In order to generate a standard formatting for the analysis, the broken channel is then *straightened* so that it is now parallel to the y-axis (Fig. 59b). For this purpose, all the coordinates of the points of the broken channel are divided by the following rotation matrix :
$$R = \begin{bmatrix} \cos\alpha_{drift} & -\sin\alpha_{drift} \\ \sin\alpha_{drift} & \cos\alpha_{drift} \end{bmatrix}$$
where $\alpha_{drift}$ is the drift angle.

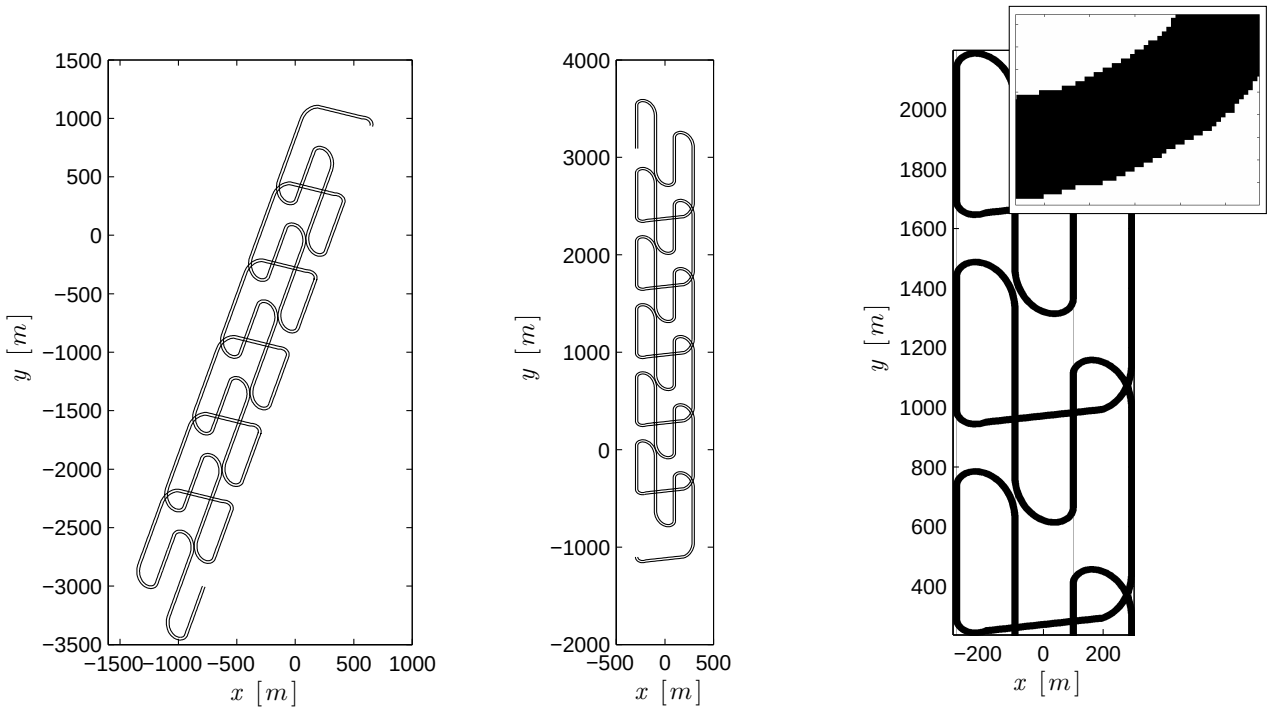
3. Then, a boolean matrix $ICE$ is created to generate a discretized version of the broken ice pattern ($true$ : ice, $false$ : broken channel), considering a discretization step $dx$. Each point of the broken channel is then assigned its closest position in the matrix $ICE$. The beginning and the end of the broken channel are also eliminated, as these parts are not representative of steady-state operations. The MatLab function `poly2mask` is then used to *fill* the area within the two sides of the broken channel (Fig. 59c).

4. The fourth step consists in the analysis of the $ICE$ matrix to determine the size of the different floes. The algorithm depends on the considered management technique.

### *4.1.2 Linear Technique*

In the case of linear technique, the ice management vessel generates stripes parallel to the drift direction. As described in Fig. 60, the $ICE$ matrix is scanned line by line, and floes are set to grow as the lines are scanned. A new floe is considered to be formed when one of the two following conditions are satisfied :

- Reading a new line does not increase its size (floes 1, 2, 3, 4, 5, 9 and 11, Fig. 60).
- The length of the floe (vertically) becomes larger than the width between two stripes (floes 6, 7, 8, 10 and 12, Fig. 60).




(a) 1. Input broken channel

(b) 2. Straightened broken channel

(c) 3. *ICE* matrix filled with all broken ice (inset : close-up)

Figure 59: Discretization of the broken channel (example for the linear technique)

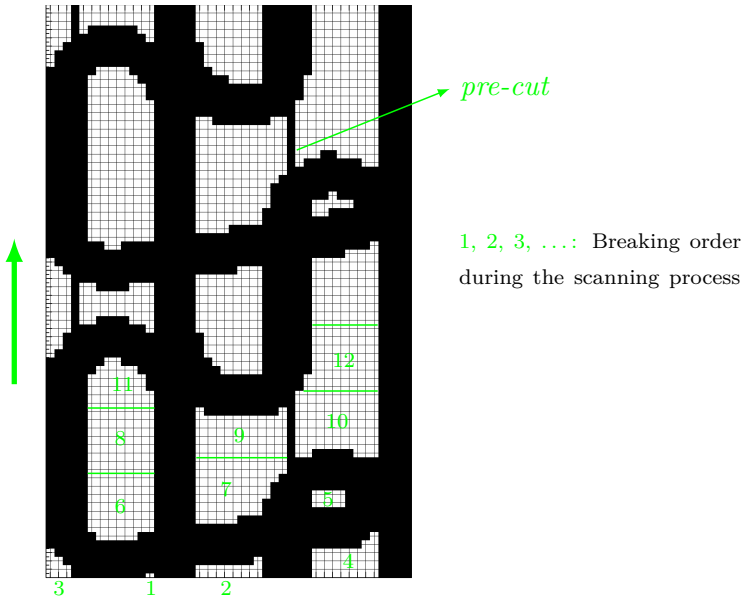



Figure 60: Linear technique : floe size detection

Note that, in some cases, additional *pre-cuts* are made in the $ICE$ matrix before the analysis (Fig. 60). They represent the cracks that will naturally form between two collinear passes of consecutive cycles.

The exact process of the matrix analysis is quite complex[33] and it would not be particularly interesting to explain this aspect in the present report. The code is however well documented and any interested reader is invited to refer to Appendix D for more information on that aspect.

### *4.1.3 Sector Technique*

The case of the sector technique is much easier to analyze. As all the stripes are perpendicular to the drift direction, their shape is almost constant and the $ICE$ matrix can be scanned column by column. Here, compared to the linear technique, no floe breaking is considered initially : first, the large stripes are detected (Fig. 61, numbers in red), and the area related to each of them is recorded column by column.

Then, in a second step, each stripe is decomposed into small floes, analyzing the area contained in each column of the matrix $ICE$. A new floe appears when

- reading a new column does not increase its size (floes 9, 17 and 25, Fig. 61); or
- the scanning process reaches the right side of the matrix[34] (floes 33, 41 and 49, Fig. 61); or
- the length of the floe (horizontally) becomes larger than the width between two stripes (floes 2 to 8 and 26 to 32 in Fig. 61, for example).

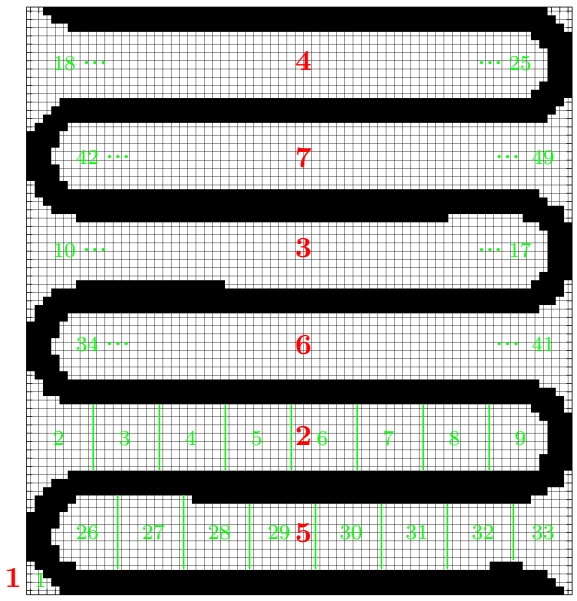
### *4.1.4 Circular Technique*

The analysis method of the $ICE$ matrix for the circular technique is quite similar to the one described for the linear technique. The main change is that the scanning is performed column by column, instead of line by line (Fig. 62), with the same floe breaking criteria as previously.

[33] Among the various difficulties, the two following examples of Fig. 60 can be given :
- the floe 9, ending with two ice pieces connected to the same floe;
- the floe 10, starting with two independent ice pieces, later connecting into one larger floe;

(the MatLab code has then been written carefully to take these two specific cases into consideration).

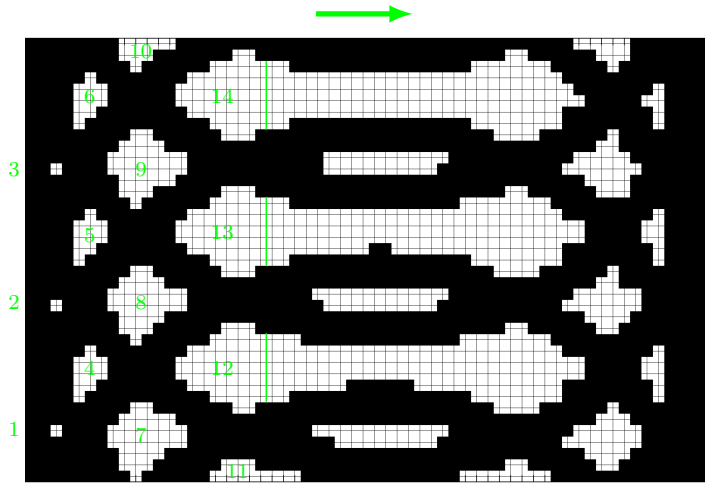
[34] Practically, these floes are still connected to the level ice, but it is assumed that they will break by themselves quite easily.




1, 2, 3, . . . : Stripes detection order during first scanning process

1, 2, 3, . . . : Breaking order of the floes during stripes scanning process

Figure 61: Sector technique : floe size detection




1, 2, 3, . . . : Breaking order during the scanning process

Figure 62: Circular technique : floe size detection

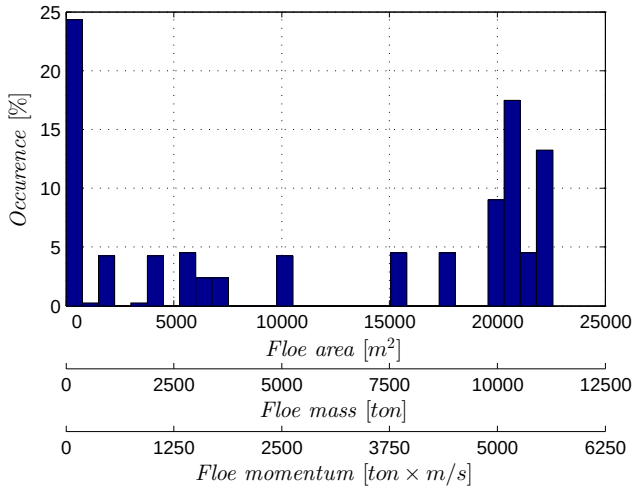
## 4.2 Outputs

In the previously described analysis of the $ICE$ matrix, the code records the size of each broken floe, so that it becomes possible to produce distribution diagrams for the area, mass and momentum of the floes :

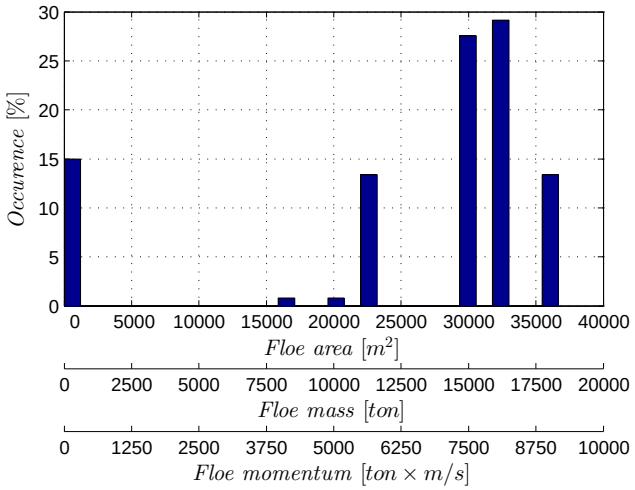
$$Floe\ area : A$$
$$Floe\ mass : M = A\left(\rho_{ice}H_{ice} + \rho_{snow}H_{snow}\right)$$
$$Floe\ momentum : M = A\left(\rho_{ice}H_{ice} + \rho_{snow}H_{snow}\right)V_{drift}$$

with $H_{ice}$, $H_{snow}$, $\rho_{ice}$ and $\rho_{snow}$ the ice and snow thicknesses and their respective densities.

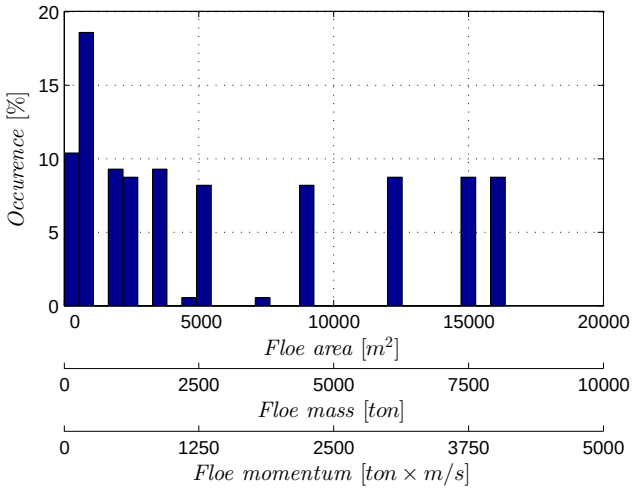
Making use of the MatLab `hist` function, the code finally generates the histogram of these distributions. Example of typical results for the three techniques are given in Fig. 63.




(a) Linear technique




(b) Sector technique




(c) Circular technique

Figure 63: Typical area, mass and momentum distributions of the three management techniques

# CONCLUSIONS AND FUTURE PROSPECTS

The objective of the present thesis was to develop a simulation tool for Ice Management Operations. This task has been completely fulfilled and the result is a complete MatLab simulation code suitable for both preliminary platform design and operations planning. Moreover, its modular structure easily allows future improvements of the code, as each module can be replaced by another.

After a short description of the ice management activities and a review of the different ice management techniques, the modular structure of the software has been presented. Each module has then been described and explained : origin of the method, transcription in MatLab code and finally validation.

The estimation of the **level ice resistance** was based on the empirical formulation of Lindqvist [6], slightly modified according to the method employed by the Hamburg Ship Model Basin. Validation has been successfully achieved through comparison with experimental data from model tests. As an extension, the **resistance in floes** has also been estimated, using an interpolation curve between the channel resistance and the level ice resistance. Influence of the ice concentration was also taken into account.

Another important module was the calculation of **ship's motion through ridges**. The procedure consisted in numerical application of theoretical models developed by Slettebø & Ueland [7] and Ehle-Myland [8] [9]. A dynamic model has been written in MatLab, so that the ship is able to use her kinetic energy to break the ridge. Again, validation has been performed from comparison with model tests results. Satisfactory simulation results have been obtained, regards to the complexity of the ridge breaking process.

The last module related to a single method was the **manoeuvrability estimation**. From a given ship's velocity and turning radius, a method has been developed to compute the required power level and pods/rudder angle. The method was based on a theoretical approach of the turning maneuver, coupled with the Lindqvist resistance estimation method in linear motion.

The next step was about the **ice management simulation** itself. Using all the modules described above, three different ice management techniques have been simulated. Starting from the structure dimensions, width and length of the managed area have been defined, and then vessel trajectory has been computed. For each technique, the required ship's velocity and power level were calculated in order for the vessel to achieve a *just-in-time* ice management, that is to say, the vessel advances at the appropriated velocity so that after a management cycle, it is in the appropriated position to start a second cycle.

Finally, a module has been written for the **analysis of the resulting floe size**. From the broken ice pattern produced by the simulation, the module discretizes the domain and generates a matrix representation of the ice. This matrix is then analyzed and distributions of the resulting floes size, mass and momentum are produced.

## FUTURE PROSPECTS

Numerous future prospects can be suggested for the present work, both at local and global scale.

At local scale, the manoeuvrability module could be improved through additional work about the empirical parameter used to fit simulation with experimental results. It might be interesting to consider how this factor is influenced by the ship's dimensions and hull geometry. Another local improvement could be a further investigation of the parameters involved in the calculation of the maximum ridge resistance. This resistance strongly depends on the value of these parameters and additional research is required in order to improve their prediction.

At larger scale, many possible improvements are related to the ice management techniques. It might be very interesting to implement a simulation code involving a ridge breaking. The simulation would be used to evaluate the requested icebreaking power to manage a drifting ridge, while still managing the surrounding level ice.

Another interesting improvement would be the implementation of a simulation code involving several icebreakers. It would then be possible to simulate multiple stage ice management, where a large icebreaker is used to break the drifting ice into very large floes, and one (or more) smaller vessels further reduce the floe size, not necessarily with the same management technique as the larger vessel. A major feature of the resulting tool would be the study of which combinations of techniques produce the most efficient ice management.

# ACKNOWLEDGEMENT

*This thesis was developed in the frame of the European Master Course in Integrated Advanced Ship Design named EMSHIP for European Education in Advanced Ship Design, Ref.: 159652-1-2009-1-BE-ERA MUNDUS-EMMC.*

# REFERENCES

[1] R. SKJETNE. *Designing for effective stationkeeping in ice.* Norwegian University of Science and Technology, 2013.

[2] B. MADDOCK, A. BUSH, and T. WOJAHN. *Advances in Ice Management for Deepwater Drilling in the Beaufort Sea.* 21st Conference on Port and Ocean Engineering under Arctic Conditions (POAC), 2011.

[3] P. DUNDERDALE and B. WRIGHT. *Pack ice management on the Southern Grand Banks offshore Newfoundland, Canada.* National Research Council of Canada, 2005.

[4] *Forecasters Handbook for the Arctic : Appendix A.* U.S. Naval Research Laboratory Marine Meteorology Division.

[5] I. SHEYKIN. *Icebreaker Reconnaissance for Ice Management : Offshore experience.* Arctic and Antarctic Research Institute of Saint-Petersburg, 2010.

[6] G. LINDQVIST. *A straightforward method for calculation of ice resistance of ships.* Report of the 10th Port and Ocean Engineering under Arctic Conditions, 1989.

[7] S. UELAND and S. SLETTEBØ. *Control Strategies for Manoeuvering in Ice Ridge and Multi Ice Regimes.* Norwegian University of Science and Technology, 2010.

[8] D. EHLE. *Analysis of Breaking through Sea Ice Ridges for Development of a Prediction Method.* Universität Duisburg-Essen, 2011.

[9] D. MYLAND. *Ships Breaking through Sea Ice Ridges.* Hamburgische Shiffbau-Versuchsanstalt, 2013.

[10] K. RISKA, M. WILHELMSON, and K. ENGLUND. *Performance of merchant vessels in ice in the Baltic.* Helsinky University of Technology, 1997.

[11] N. REIMER. *Ice Route Optimisation.* The Ice Class Ships - Royal Institution of Naval Architects, 2012.

[12] K.V. HOYLAND et al. *Physical Modeling of First-Year Ice Ridges - Part 1 : Production, Consolidation and Physical Properties.* Norwegian University of Science and Technology, Hamburgische Shiffbau-Versuchsanstalt, 2011.

[13] M. MELLOR. *Ship resistance in thick brash ice.* Cold Regions Science and Technology 3, Elsevier, 1980.

[14] J.M. HAMILTON, C.J. HOLUB, and J. BLUNT. *Simulation of Ice Management Fleet Operations using Two Decades of Beaufort Sea Ice Drift and Thickness Time Histories.* International Offshore and Polar Engineering Conference, 2011.

[15] C. DALEY and K. RISKA. *Conceptual Framework for an Ice Load Model.* National Energy Board of Canada.

[16] I. FROLOV et al. *Ice Management : From the Concept to Realization*. Arctic and Antarctic Research Institute of Saint-Petersburg, 2012.

[17] J.M. HAMILTON, C.J. HOLUB, and J. BLUNT. *Ice Management for Support of Arctic Floating Operations*. Arctic Technology Conference, 2011.

[18] J.M. HAMILTON, D.M. FENZ, T. KOKKINIS, and C.J. HOLUB. *Aligning the needs of floating drilling and the capabilities of ice management*. 22nd Conference on Port and Ocean Engineering under Arctic Conditions (POAC), 2013.

[19] J. LIU, M. LAU, and F.M. WILLIAMS. *Mathematical Modeling of Ice-Hull Interaction for Ship Maneuvering in Ice Simulations*. International Conference and Exhibition on Performance of Ships and Structures in Ice (IceTech), 2006.

# APPENDICES

## A VALIDATION DATA

### A.1 Level Ice Resistance

#### *A.1.1 Reference Ships Data*

Table 5: Reference ships dimensions for validation of the level ice resistance estimation

| Vessel | | Ship n$^o$1 | Ship n$^o$2 | Ship n$^o$3 |
|---|---|---|---|---|
| $L_{pp}$ | $[m]$ | 184.3 | 126.6 | 72.9 |
| $B$ | $[m]$ | 28.9 | 23 | 18.4 |
| $T$ | $[m]$ | 9.5 | 7.5 | 6.52 |
| $L_{bow}$ | $[m]$ | 21.39 | 46.8 | NA |
| $L_{mid}$ | $[m]$ | 152.6 | 82.06 | NA |
| $L_{wedge}$ | $[m]$ | 24.5 | 11.6 | 0 |
| $\alpha_{wedge}^{max}$ | $[deg]$ | 25 | 25 | 0 |
| $\phi_0$, $\alpha_0$, $\psi_0$ | $[deg]$ | [22, 31, 38.1] | [24, 85, 24.1] | [24, 32, 40.0] |
| $\phi_{1/4}$, $\alpha_{1/4}$, $\psi_{1/4}$ | $[deg]$ | [25, 39, 36.5] | [22, 29, 39.8] | [26, 45, 34.6] |
| $\phi_{2/4}$, $\alpha_{2/4}$, $\psi_{2/4}$ | $[deg]$ | [24, 42, 33.6] | [31, 23, 57.0] | [27, 41, 37.8] |
| $\phi_{3/4}$, $\alpha_{3/4}$, $\psi_{3/4}$ | $[deg]$ | [24, 43, 33.1] | [36, 17, 68.1] | [27, 41, 37.8] |
| $\phi_{15/16}$, $\alpha_{15/16}$, $\psi_{15/16}$ | $[deg]$ | [23, 26, 44.0] | [32, 11, 73.7] | [27, 24, 51.4] |
| $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ | $[deg]$ | [23.6, 36.2, 37.1] | [29.0, 32.9, 52.54] | [26.2, 36.6, 40.3] |
| $\mu$ | $[-]$ | 0.1 | 0.1 | 0.1 |
| $m_{ship}$ | $[ton]$ | 38945 | 15555 | NA |
| $m_{add}/m_{ship}$ | $[-]$ | 0.06 | 0.06 | NA |

### A.2 Moving through Ice Ridges

#### *A.2.1 Reference Ships Data*

Table 6: Reference ships dimensions for validation of the ridge breaking resistance estimation

| Vessel | | Ship n$^o$1 | Ship n$^o$2 |
|---|---|---|---|
| $L_{pp}$ | $[m]$ | 184.3 | 126.6 |
| $B$ | $[m]$ | 28.9 | 23 |
| $T$ | $[m]$ | 9.5 | 7.5 |
| $L_{bow}$ | $[m]$ | 21.39 | 46.8 |
| $L_{mid}$ | $[m]$ | 152.6 | 82.06 |
| $L_{wedge}$ | $[m]$ | 24.5 | 11.6 |
| $\alpha_{wedge}^{max}$ | $[deg]$ | 25 | 25 |
| $\phi_0$, $\alpha_0$, $\psi_0$ | $[deg]$ | [22, 31, 38.1] | [24, 85, 24.1] |
| $\phi_{1/4}$, $\alpha_{1/4}$, $\psi_{1/4}$ | $[deg]$ | [25, 39, 36.5] | [22, 29, 39.8] |
| $\phi_{2/4}$, $\alpha_{2/4}$, $\psi_{2/4}$ | $[deg]$ | [24, 42, 33.6] | [31, 23, 57.0] |
| $\phi_{3/4}$, $\alpha_{3/4}$, $\psi_{3/4}$ | $[deg]$ | [24, 43, 33.1] | [36, 17, 68.1] |
| $\phi_{15/16}$, $\alpha_{15/16}$, $\psi_{15/16}$ | $[deg]$ | [23, 26, 44.0] | [32, 11, 73.7] |
| $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ | $[deg]$ | [23.6, 36.2, 37.1] | [29.0, 32.9, 52.54] |
| $\mu$ | $[-]$ | 0.1 | 0.1 |
| $m_{ship}$ | $[ton]$ | 38945 | 15555 |
| $m_{add}/m_{ship}$ | $[-]$ | 0.06 | 0.06 |

### *A.2.2 Validation Data*

Table 7: Ridge breaking resistance : validation data

| Validation test | | A | B | C | D |
|---|---|---|---|---|---|
| Vessel | | Ship n$^o$2 | Ship n$^o$2 | Ship n$^o$2 | Ship n$^o$1 |
| $L_{ridge}$ | $[m]$ | 42.0 | 30.0 | 42.0 | 40 |
| $H_k$ | $[m]$ | 7.6 | 6.9 | 7.6 | 7.5 |
| $k$ | $[-]$ | 1.75 | 1.75 | 1.75 | 1.75 |
| $N$ | $[-]$ | 0.5 | 0.5 | 0.5 | 0.5 |
| $\phi_s$ | $[deg]$ | 57 | 57 | 57 | 57 |
| $\eta$ | $[-]$ | 0.5 | 0.5 | 0.5 | 0.42 |
| $L_{CD}$ | $[m]$ | 40 | 150 | 120 | 50 |
| $L_{HI}$ | $[m]$ | 200 | 250 | 250 | 250 |
| $H_{ice}$ | $[m]$ | 0.9 | 0.95 | 0.9 | 1.25 |
| $H_{snow}$ | $[m]$ | 0 | 0 | 0 | 0 |
| $\sigma_b$ | $[kPa]$ | 1100 | 1000 | 1100 | 1450 |

### *A.2.3 From Model Scale to Full Scale Data*

Table 8: Scaling rules according to Froude similitude

| Quantity | Scaling factor |
|---|---|
| Lengths | $\lambda$ |
| Angles | 1 |
| Displacement | $\lambda^3$ |
| Position | $\lambda$ |
| Velocity | $\lambda^{1/2}$ |
| Acceleration | 1 |
| Power | $\lambda^{7/2}$ |
| Thrust, Resistance | $\lambda^3$ |
| Revolution rate | $\lambda^{-1/2}$ |
| Time | $\lambda^{1/2}$ |
| Density | 1 |
| Ice thickness | $\lambda$ |
| Bending strength | $\lambda$ |
| Young's modulus | $\lambda$ |

## A.3 Manoeuvrability Estimation

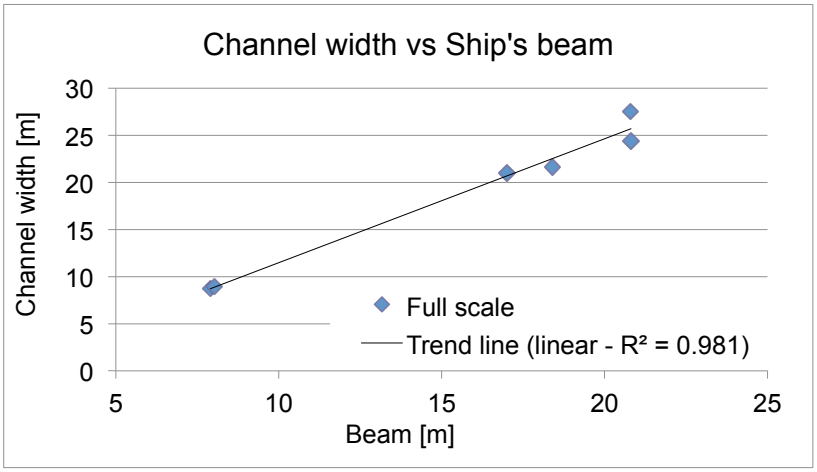
### *A.3.1 Reference Ships Data*

Table 9: Reference ships dimensions for validation of the manoeuverability estimation

| Vessel | | Research vessel | Emergency vessel | Inland icebreaker |
|---|---|---|---|---|
| $L_{pp}$ | $[m]$ | 85.9 | 72.6 | 32.6 |
| $B$ | $[m]$ | 21.6 | 18.4 | 8.03 |
| $T$ | $[m]$ | 8.0 | 6.52 | 1.60 |
| $L_{bow}$ | $[m]$ | 17.2 | 10.9 | 9.8 |
| $L_{mid}$ | $[m]$ | 55.8 | 54.5 | 14.7 |
| $L_{wedge}$ | $[m]$ | 16.8 | 0 | 0 |
| $\alpha_{wedge}^{max}$ | $[deg]$ | 16 | 0 | 0 |
| $\phi_0$, $\alpha_0$, $\psi_0$ | $[deg]$ | [20, 46, 26.8] | [24, 32, 40.0] | [19, 31, 33.8] |
| $\phi_{1/4}$, $\alpha_{1/4}$, $\psi_{1/4}$ | $[deg]$ | [24, 40, 34.7] | [26, 45, 34.6] | [22, 28, 40.7] |
| $\phi_{2/4}$, $\alpha_{2/4}$, $\psi_{2/4}$ | $[deg]$ | [28, 38, 40.8] | [27, 41, 37.8] | [25, 23, 50.0] |
| $\phi_{3/4}$, $\alpha_{3/4}$, $\psi_{3/4}$ | $[deg]$ | [29, 30, 48.0] | [27, 41, 37.8] | [24, 16, 58.2] |
| $\phi_{15/16}$, $\alpha_{15/16}$, $\psi_{15/16}$ | $[deg]$ | [27.5, 14, 65.0] | [27, 24, 51.4] | [16, 7, 67.0] |
| $\bar{\phi}$, $\bar{\alpha}$, $\bar{\psi}$ | $[deg]$ | [25.7, 33.6, 43.1] | [26.2, 36.6, 40.3] | [21.2, 21, 50.0] |
| $\mu$ | $[-]$ | 0.1 | 0.1 | 0.1 |
| $m_{ship}$ | $[ton]$ | 9850 | 6027 | 233.2 |
| $m_{add}/m_{ship}$ | $[-]$ | 0.06 | 0.06 | 0.06 |
| *Propulsion* | $[-]$ | Dual pod, $\phi = 4\,m$ $5\,m$ from centreline | Dual pod, $\phi = 3.5\,m$ $4.5\,m$ from centreline | Single screw $\phi = 1.5\,m$ + Rudder |

## A.4 Width of the Broken Channel

Table 10: Reference data for the estimation of the broken channel width of an icebreaker




| | | Full scale | | Model scale | |
|---|---|---|---|---|---|
| **Ship number** | **Scale factor** | **Ship's beam** | **Channel width** | **Channel width** | **Std. dev.** |
| | [-] | [m] | [m] | [m] | [m] |
| Ship 1 | 17.30 | 17.00 | 21.02 | 1.22 | 0.07 |
| Ship 2 | 6.000 | 7.90 | 8.74 | 1.46 | - |
| Ship 3 | 7.500 | 8.03 | 8.93 | 1.19 | - |
| Ship 4 | 18.919 | 18.40 | 21.64 | 1.14 | - |
| Ship 5 | 18.600 | 20.81 | 24.37 | 1.31 | 0.10 |
| Ship 6 | 17.143 | 20.80 | 27.51 | 1.61 | 0.13 |

## B RIDGE MAXIMUM RESISTANCE CALCULATION

According to Mellor [13], the maximum of total resistance can be calculated by summing the resistance due to the ship's bow with the resistance due to the friction on the parallel midbody :

$$R_{ridge}^{max} = R_{bow} + R_{mid}$$

**Bow resistance** The bow resistance $R_{bow}$ has been estimated by Ueland and Slettebø [7] as the sum of the crushing resistance and the frictional resistance on the bow :

$$R_{bow} = R_{bow\,crushing} + R_{bow\,friction} = BR + 2\mu L_{bow} R$$

In this expression,

- $B$ is the ship's breadth;
- $R$ is the ice yield resistance, its definition and calculation method will be given further below;
- $\mu$ is the friction coefficient of the ice on the hull paint;
- $L_{bow}$ is the ship's bow length.

**Midbody resistance** The midbody resistance is assumed to consist only of frictional resistance of the midbody on the sides of the broken ridge. It is estimated by

$$R_{mid} = 2\mu R N L_{mid}$$

where $L_{mid}$ is the ship's parallel midbody length and where the factor $N$ is still an important unknown of the problem. Following Mellor [13] and Ueland and Slettebø [7], the following wide range of values can be given :

$$N \in [0.06, 0.50]$$

**Yield resistance** The yield resistance $R$ used in the previous expressions is defined as the normal force on a vertical strip element of the wedge that appears when the ship's bow penetrates the ice. Mellor assumed this configuration to be equivalent to a vertical plate horizontally pushed against a homogenous layer of brash ice[35] (Fig. 64). He developed the following expression for the yield resistance :

$$R = \frac{1}{2}\frac{1+\sin\phi_s}{1-\sin\phi_s}(1-\eta)\rho_i g\left(1-\frac{\rho_i}{\rho_w}\right)H_k^2$$

with

- $\phi_s$ the inner friction angle;
- $\eta$ the ice porosity;
- $\rho_i$ and $\rho_w$ respectively the ice and water densities;
- $H_k$ the ridge keel depth.

[35] The main hypothesis of Mellor is to assumed that ridges have almost the same characteristics as compacted thick brash ice.

In this expression, the ice porosity is generally around 40−50%. However, the angle $\phi_s$ is the main source of uncertainty of the problem, even more than the factor $N$ previously defined. According to Mellor, it is comprised between $42^o$ and $58^o$, so that the factor $(1+\sin\phi_s)/(1-\sin\phi_s)$ varies between 5.04 and 12.16 !

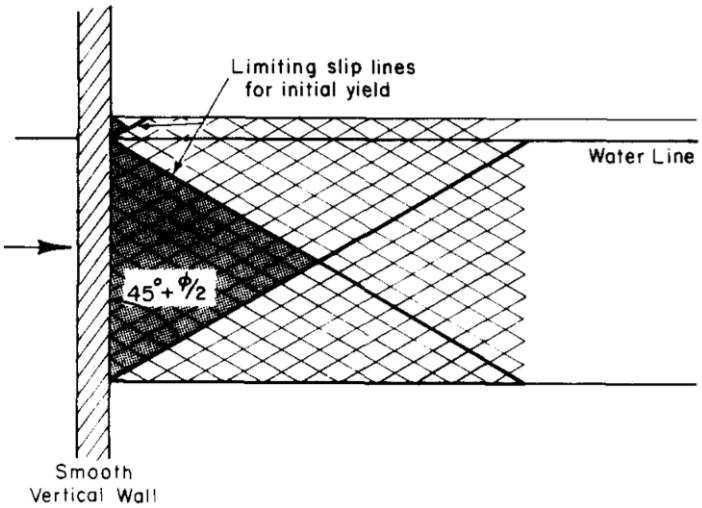



Figure 64: Schematic slip lines for cohesionless brash ice pushed by a wide and smooth vertical wall [13]

**Improvements by Myland [9]** From the ridge breaking experiments carried by Myland in 2010-2011, it has been noticed that the ridge breaking resistance depends on the penetration velocity $v_{pen}$ (i.e. the ship's velocity at the instant of bow impact). Considering this new information, Myland introduced the dimensionless factor $\frac{kv_{pen}}{\sqrt{gH_k}}$ (ice Froude number) and suggests the following improved formula :

$$R_{ridge}^{max} = \frac{kv_{pen}}{\sqrt{gH_k}}(BR + 2\mu L_{bow}R) + 0.63\frac{kv_{pen}}{\sqrt{gH_k}}(2\mu RNL_{mid})$$

In this expression, the factor $k$ depends on the ship ridge breaking capability and the number of rams required to break the ridge :

- $k \in [1.1, 1.75]$ if the ships breaks the ridge at the first advance (no ramming)
- $k \in [1.3, 2.0]$ if ramming is required

In the present work, the ramming mode is not considered, so that only the first range is to be used.

## C RIDGE RESISTANCE VALIDATION : ADDITIONAL CASES

Additionally to validation tests $A$ and $D$ presented in section II.2.4.4, we give here the results for cases $B$ and $C$ (Fig. 65 and 66).

Test runs $B$ and $C$ are relatively similar. In both cases, the measured ice resistance shows large variations, probably due to the successive individual breakings of ice cusps. The thrust is initially slightly overestimated, but the final equilibrium velocity in level ice is close to the experimental value. Compared to test run $A$, the velocity drop is here better modeled. In test run $C$, the ship recovers however her velocity much faster than during the experiment. Again, the acceleration is correctly modeled in both tests during the whole process.

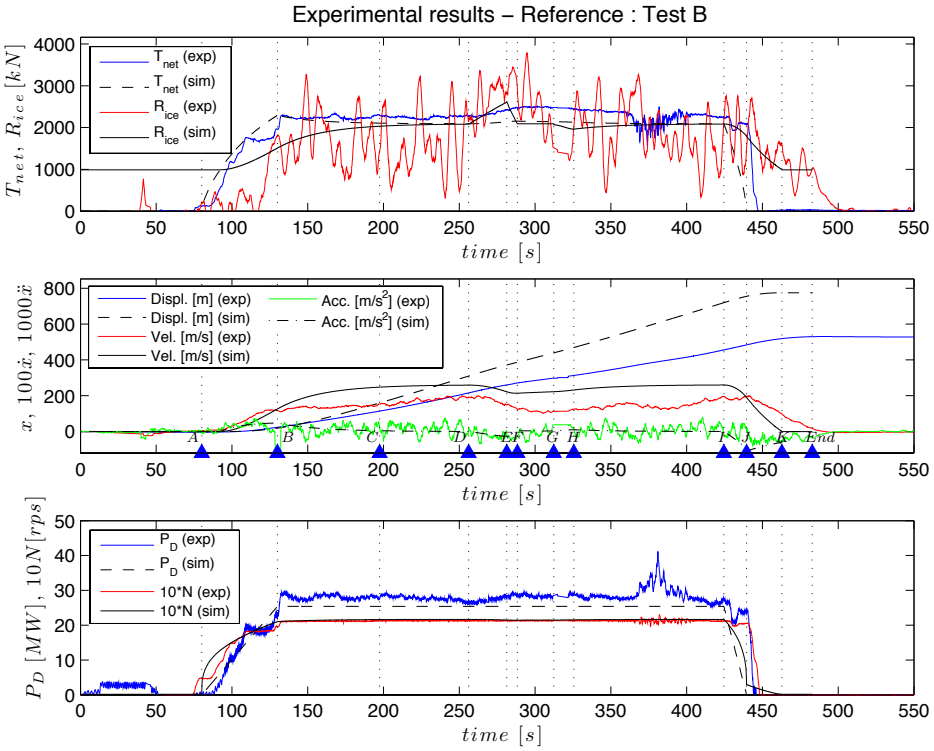



Figure 65: Ridge breaking validation : test B

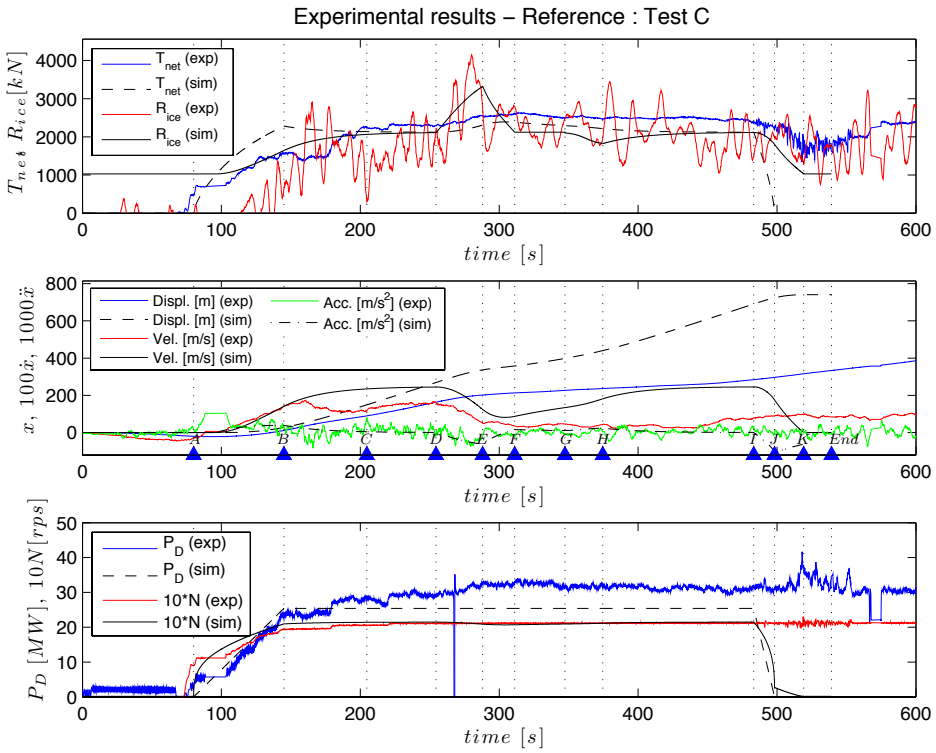



Figure 66: Ridge breaking validation : test C

# D MATLAB CODE

## D.1 Inputs Reading

### *D.1.1 Input_reading.m*

```
function Input_reading
%
% Reads all parameters input files and generate global variables
% containing the corresponding input data.
%

global Ice_Water_data
global Ship_data
global Prop_data
global Ridge_data
global Structure_data
global Ice_drift_data
global Other_data

[Ice_Water_data] = Read_Ice_Water_parameters;
[Ship_data]      = Read_Ship_parameters;
[Prop_data]      = Read_Propeller_data;
[Ridge_data]     = Read_Ridge_parameters;
[Structure_data] = Read_Structure_parameters;
[Ice_drift_data] = Read_Ice_drift_data;
[Other_data]     = Read_Other_parameters;

end
```

### *D.1.2 Read_Ice_drift_data.m*

```
function [Ice_drift_data] = Read_Ice_drift_data
%
% Reading the input parameter file "Ice_drift_data.txt"
%

fid = fopen('Input_data/Ice_drift_data.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t')');
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref     = line{4};
            case 'Drift_velocity',
                V_drift = str2double(line{4});
            case 'Drift_direction_clockwise',
                alpha_drift = str2double(line{4});
            case 'Drift_direction_change_rate',
                alpha_drift_dot_max = str2double(line{4});
        end;
    end;
end;

fclose(fid);


Ice_drift_data = {Ref; V_drift; alpha_drift; alpha_drift_dot_max};

end
```

### *D.1.3 Read_Ridge_parameters.m*

```
function [Ridge_data] = Read_Ridge_parameters
%
% Reading the input parameter file "Ridge_parameters.txt"
%

fid = fopen('Input_data/Ridge_parameters.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t')');
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref     = line{4};
            case 'Length_ridge',
                L_ridge  = str2double(line{4});
            case 'Keel_depth_ridge',
                H_K  = str2double(line{4});
            case 'Distance_travelled_at_full_speed_before_ridge',
                L_adv_CD = str2double(line{4});
            case 'Distance_travelled_at_full_power_after_ridge',
                L_adv_HI = str2double(line{4});
            case 'Coeff_k_for_max_ridge_resistance',
                k_ridge  = str2double(line{4});
            case 'Coeff_N_for_max_ridge_resistance',
                N_ridge  = str2double(line{4});
            case 'Inner_friction_angle_for_yield_resistance',
                phi_s  = str2double(line{4});
            case 'Ice_porosity_in_ridge',
                eta_ridge = str2double(line{4});
        end;
    end;
end;

fclose(fid);

Ridge_data = {Ref; L_ridge; H_K; L_adv_CD; L_adv_HI; k_ridge;...
                N_ridge; phi_s; eta_ridge};
end
```

### *D.1.4 Read_Structure_parameters.m*

```
function [Structure_data] = Read_Structure_parameters
%
% Reading the input parameter file "Structure_parameters.txt"
%

fid = fopen('Input_data/Structure_parameters.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t')');
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref     = line{4};
            case 'Length',
                L_struct = str2double(line{4});
            case 'Breadth',
                B_struct = str2double(line{4});
            case 'Diameter',
                D_struct = str2double(line{4});
            case 'Course_clockwise',
                Psi_struct   = str2double(line{4});
            case 'Critical_distance_to_level_ice',
                t_cr    = str2double(line{4});
            case 'Ships_exclusion_zone_radius',
                R_excl    = str2double(line{4});
            case 'Target_floe_size',
                d_floe_target = str2double(line{4});
            case 'Double_pass_technique',
                if strcmp(line{4},'true') || strcmp(line{4},'True'),
                    double_pass = true;
                elseif strcmp(line{4},'false') || strcmp(line{4},'False'),
                    double_pass = false;
                else
                    error(['Unable to interpret double_pass value in ',...
                        'Structure_parameters.txt. Please use "true"',...
                        'or "false" values.'])
                end;
        end;
    end;
end;

fclose(fid);

Struct_dimensions = [L_struct, B_struct, D_struct];
Structure_data = {Ref; Struct_dimensions; Psi_struct; t_cr; R_excl;...
                  d_floe_target; double_pass};

end
```

### *D.1.5 Read_Ice_Water_parameters.m*

```
function [Ice_Water_data] = Read_Ice_Water_parameters
%
% Reading the input parameter file "Ice_Water_parameters.txt"
%

fid = fopen('Input_data/Ice_Water_parameters.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t')');
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref = line{4};
            case 'Ice_thickness',
                H_ice   = str2double(line{4});
            case 'Snow_thickness',
                H_snow   = str2double(line{4});
            case 'Ratio_H_snow_eff/H_snow',
                H_snow_eff_H_snow = str2double(line{4});
            case 'Bending_strength',
                sigma_b_kPa  = str2double(line{4});
            case 'Water_density',
                rho_w   = str2double(line{4});
            case 'Ice_density',
                rho_i   = str2double(line{4});
            case 'Snow_density',
                rho_s   = str2double(line{4});
            case 'Elastic_modulus',
                E_kPa   = str2double(line{4});
            case 'Poisson_ratio',
                nu    = str2double(line{4});
            case 'Coeff_k_breaking_resistance',
                k   = str2double(line{4});
            case 'Ratio_L_characteristic/L_cusp',
                Lch_Lcusp = str2double(line{4});
            case 'Ship_bottom_ice_coverage',
                Cov_B   = str2double(line{4});
            case 'Propeller_milling_ice_thickness',
                H_milling = str2double(line{4});
        end;
    end;
end;

fclose(fid);


rho         = [rho_w, rho_i, rho_s];
H_snow_vec = [H_snow, H_snow_eff_H_snow];

Ice_Water_data = {Ref; H_ice; H_snow_vec; sigma_b_kPa; rho; E_kPa;...
                  nu; k; Lch_Lcusp; Cov_B; H_milling};

end
```

### *D.1.6 Read_Ship_parameters.m*

```
function [Ship_data] = Read_Ship_parameters
%
% Reading the input parameter file "Ship_parameters.txt"
%

fid = fopen('Input_data/Ship_parameters.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t'));
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref = line{4};
            case 'Length',
                L  = str2double(line{4});
            case 'Breadth',
                B  = str2double(line{4});
            case 'Draught',
                T  = str2double(line{4});
            case 'Bow_length',
                L_bow = str2double(line{4});
            case 'Midbody_length',
                L_mid  = str2double(line{4});
            case 'Wedge_Length',
                LWEDGE   = str2double(line{4});
            case 'Max_Wedge_Angle',
                ALPHWED  = str2double(line{4});
            case 'Stem_Angle_0',
                PHI_0  = str2double(line{4});
            case 'WL_Angle_0',
                ALPHA_0  = str2double(line{4});
            case 'Normal_Angle_0',
                PSI_0  = str2double(line{4});
            case 'Buttock_Angle_1_4',
                PHI_1_4  = str2double(line{4});
            case 'WL_Angle_1_4',
                ALPHA_1_4  = str2double(line{4});
            case 'Normal_Angle_1_4',
                PSI_1_4  = str2double(line{4});
            case 'Buttock_Angle_2_4',
                PHI_2_4  = str2double(line{4});
            case 'WL_Angle_2_4',
                ALPHA_2_4  = str2double(line{4});
            case 'Normal_Angle_2_4',
                PSI_2_4  = str2double(line{4});
            case 'Buttock_Angle_3_4',
                PHI_3_4  = str2double(line{4});
            case 'WL_Angle_3_4',
                ALPHA_3_4  = str2double(line{4});
            case 'Normal_Angle_3_4',
                PSI_3_4  = str2double(line{4});
            case 'Buttock_Angle_15_16',
                PHI_15_16  = str2double(line{4});
            case 'WL_Angle_15_16',
                ALPHA_15_16  = str2double(line{4});
            case 'Normal_Angle_15_16',
                PSI_15_16  = str2double(line{4});
            case 'Mean_Buttock_Angle',
                PHIm   = str2double(line{4});
            case 'Mean_WL-Angle',
                ALPHAm   = str2double(line{4});
            case 'Mean_Normal_Angle_new',
                PSIm   = str2double(line{4});
            case 'x_coord_WL_from_aft_pp_0',
                x_0  = str2double(line{4});
            case 'x_coord_WL_from_aft_pp_1_4',
                x_1_4  = str2double(line{4});
            case 'x_coord_WL_from_aft_pp_2_4',
                x_2_4  = str2double(line{4});
            case 'x_coord_WL_from_aft_pp_3_4',
                x_3_4  = str2double(line{4});
            case 'x_coord_WL_from_aft_pp_15_16',
                x_15_16  = str2double(line{4});
            case 'Friction_Coefficient',
                MU   = str2double(line{4});
            case 'Channel_width_to_beam_ratio',
                k_channel = str2double(line{4});
            case 'Turning_calc_coeff',
                k_turning = str2double(line{4});
            case 'Ship_displacement',
                m_ship_ton = str2double(line{4});
            case 'Added_mass_coeff_x',
                m_add_coeff_x = str2double(line{4});
            case 'Max_delivered_power_MW',
                P_D_max_MW = str2double(line{4});
            case 'time_from_0_to_P_D_max',
                time_to_P_D_max = str2double(line{4});
            case 'time_from_P_D_max_to_0',
                time_P_D_max_to_0 = str2double(line{4});
            case 'Propeller_mode_turning',
                prop_mode = str2double(line{4});
            case 'Advance_mode_trajectory',
                advance_mode = str2double(line{4});
            case 'Fixed_prop_and_rudder',
                rudder = str2double(line{4});
            case 'V_unit_m/s_or_kts',
                V_unit_ow = line{4};
            case 'Min_velocity_ow',
                V_min_ow = str2double(line{4});
            case 'coeff_ow_a',
                coeff_ow_a = str2double(line{4});
            case 'coeff_ow_b',
                coeff_ow_b = str2double(line{4});
            case 'coeff_ow_c',
                coeff_ow_c = str2double(line{4});
            case 'coeff_ow_d',
                coeff_ow_d = str2double(line{4});
            case 'coeff_ow_e',
                coeff_ow_e = str2double(line{4});
            case 'coeff_ow_f',
                coeff_ow_f = str2double(line{4});
            case 'coeff_ow_g',
                coeff_ow_g = str2double(line{4});
            case 'x_coordinates_hull',
                X_hull = zeros(1,length(line)-3);
                for i=1:length(X_hull),
                    X_hull(i)=str2double(line{i+3});
                end;
            case 'y_coordinates_hull',
                Y_hull = zeros(1,length(line)-3);
                for i=1:length(Y_hull),
                    Y_hull(i)=str2double(line{i+3});
                end;
        end;
    end;
end;

fclose(fid);

Lengths      = [L, L_bow, L_mid];
wedge_geom   = [LWEDGE, ALPHWED];
angles_0     = [PHI_0, ALPHA_0, PSI_0];
angles_1_4   = [PHI_1_4, ALPHA_1_4, PSI_1_4];
angles_2_4   = [PHI_2_4, ALPHA_2_4, PSI_2_4];
angles_3_4   = [PHI_3_4, ALPHA_3_4, PSI_3_4];
angles_15_16= [PHI_15_16, ALPHA_15_16, PSI_15_16];
angles_mean  = [PHIm, ALPHAm, PSIm];
masses       = [m_ship_ton, m_add_coeff_x];
power        = [P_D_max_MW, time_to_P_D_max, time_P_D_max_to_0];
coeff_ow     = [coeff_ow_a, coeff_ow_b, coeff_ow_c, coeff_ow_d,...
                coeff_ow_e, coeff_ow_f, coeff_ow_g];
X_WL_bow     = [x_0, x_1_4, x_2_4, x_3_4, x_15_16];
param_traj   = [prop_mode, advance_mode, rudder];

Ship_data = {Ref; Lengths; B; T; wedge_geom; angles_0; angles_1_4;...
             angles_2_4; angles_3_4; angles_15_16; angles_mean; MU;...
             masses; power; V_unit_ow; V_min_ow; coeff_ow; k_channel;...
             X_WL_bow; k_turning; param_traj};

end
```

### *D.1.7 Read_Propeller_data.m*

```
function [Prop_data] = Read_Propeller_data
%
% Reading the input parameter file "Propeller_data.txt"
%

fid = fopen('Input_data/Propeller_data.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t'));
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref = line{4};
            case 'Diameter',
                D = str2double(line{4});
            case 'Propellers_nbr',
                Nbr = str2double(line{4});
            case 'Wake_coeff_method',
                w_method = str2double(line{4});
            case 'Wake_coeff_w1',
                w1 = str2double(line{4});
            case 'Wake_coeff_w2',
                w2 = str2double(line{4});
            case 'Wake_coeff_w3',
                w3 = str2double(line{4});
            case 'Thdf_coeff1',
                thdf_1 = str2double(line{4});
            case 'Thdf_coeff2',
                thdf_2 = str2double(line{4});
            case 'Thdf_coeff3',
                thdf_3 = str2double(line{4});
            case 'Prop_x_positions_from_aft_pp',
                x_prop = zeros(1,length(line)-3);
                for i=1:length(x_prop),
                    x_prop(i)=str2double(line{i+3});
                end;
            case 'Prop_y_positions_from_aft_pp',
                y_prop = zeros(1,length(line)-3);
                for i=1:length(y_prop),
                    y_prop(i)=str2double(line{i+3});
                end;
            case 'Advance_number',
                J = zeros(1,length(line)-3);
                for i=1:length(J),
                    J(i)=str2double(line{i+3});
                end;
            case 'Torque_coeff',
                KQ10 = zeros(1,length(line)-3);
                for i=1:length(KQ10),
                    KQ10(i)=str2double(line{i+3});
                end;
            case 'Thrust_coeff',
                KT = zeros(1,length(line)-3);
                for i=1:length(KT),
                    KT(i)=str2double(line{i+3});
                end;
        end;
    end;
end;
fclose(fid);

w_coeff     = [w1, w2, w3, w_method];
thdf_coeff  = [thdf_1, thdf_2, thdf_3];
xy_prop_aft = [x_prop; y_prop];
Prop_data   = {Ref; D; Nbr; w_coeff; thdf_coeff; J; KQ10; KT;...
               xy_prop_aft};

end
```

### *D.1.8 Read_Other_parameters.m*

```
function [Other_data] = Read_Other_parameters
%
% Reading the input parameter file "Other_parameters.txt"
%

fid = fopen('Input_data/Other_parameters.txt', 'r');

while ~feof(fid)
    Line = fgets(fid);
    line = strread(Line,'%s','delimiter',sprintf('\t'));
    line(strcmp('',line)) = []; % Remove empty elements of the line
    if not(isempty(line)),
        switch line{1},
            case 'Reference',
                Ref = line{4};
            case 'Gravitational_constant',
                g  = str2double(line{4});
            case 'Time_step',
                dt = str2double(line{4});
            case 'Spatial_step',
                dx_ice = str2double(line{4});
        end;
    end;
end;

fclose(fid);

Other_data = {Ref; g; dt; dx_ice};

end
```

## D.2 Thrust Calculation

### *D.2.1 Thrust_calc.m*

```
function [T_net, N_calc] = Thrust_calc(P_D, V, fig)
%
% Calculation of the total net thrust of a ship, based on the propeller
% open-water characteristics.
%
% INPUTS :
% V                 [m/s]   Ship's velocity
% P_D               [W]     Total delivered power
% fig               [Bool]  Activate/Deactivate figures plotting
%
% OUTPUTS:
% T_net             [N]     Total net thrust
% N_calc            [1/s]   Revolution rate
%

global Ice_Water_data
global Ship_data
global Prop_data

% READING ICE WATER PARAMETERS
rho_w       = Ice_Water_data{5}(1); % Water density
H_milling   = Ice_Water_data{11};   % Propeller milling ice thickness

% READING SHIP PARAMETERS
L_pp        = Ship_data{2}(1);      % Ship's length

% READING INPUT PROPELLER DATA
Ref         = Prop_data{1};         % Propeller reference
D           = Prop_data{2};         % Propeller(s) diameter
Nbr         = Prop_data{3};         % Number of propellers
w_coeff     = Prop_data{4};         % Wake coefficients
thdf_coeff  = Prop_data{5};         % Thrust deduction factor coefficients
J           = Prop_data{6};         % Advance number
KQ10        = Prop_data{7};         % Torque coefficient (x10)
KT          = Prop_data{8};         % Thrust coefficient
ETA0        = KT.*J./(KQ10/10*2*pi);% Propeller efficiency


% TRENDLINES CALCULATION
[P_KQ10,S_KQ10] = polyfit(J,KQ10,4);
[P_KT,S_KT]     = polyfit(J,KT,4);

R2_KQ10     = 1 - S_KQ10.normr^2 / norm(KQ10-mean(KQ10))^2;
R2_KT       = 1 - S_KT.normr^2 / norm(KT-mean(KT))^2;
if (R2_KQ10 < 0.95) || (R2_KT < 0.95),
    disp(['WARNING : Open-water characteristics poorly estimated. ',...
          'Please run Thrust_calc_1prop.m with fig=true to visualize ',...
          'the estimation.'])
end;

% REPRESENTATION OF PROPELLER KT-KQ-ETA0 VS J DIAGRAM
if fig,
    J_repr = 0:.01:(max(J)+.2);
    trend_KQ10 = polyval(P_KQ10, J_repr);
    trend_KT   = polyval(P_KT, J_repr);
    trend_ETA0 = interp1(J, ETA0, J_repr);

    figure('Position', [0 0 900 500]),
    plot(J, KQ10, 'o', 'Color', 'b', 'MarkerFaceColor', 'b');
    hold on,
    plot(J_repr, trend_KQ10, 'b');
    hold on,
    plot(J, KT, 's', 'Color', 'r', 'MarkerFaceColor', 'r');
    hold on,
    plot(J_repr, trend_KT, 'r');
    hold on,
    plot(J, ETA0, '^', 'Color', 'g', 'MarkerFaceColor', 'g');
    hold on,
    plot(J_repr, trend_ETA0, 'g');
    legend('10*KQ', '10*KQ (trend line)', 'KT', 'KT (trend line)',...
                                                'ETA0', 'ETA0 (interp.)');
    xlabel('J [-]', 'Fontsize', 12)
    ylabel('10*KQ, KT [-]', 'Fontsize', 12)
    title(['Propeller diagram - Reference : ', Ref], 'Fontsize', 12)
    axis([0 (max(J)+.2) 0 (max([max(KQ10) max(KT) max(ETA0)])+0.1)])
end;

% THRUST CALCULATION
if V < 1e-10,
    V = 1e-10; % To avoid stability problems
end;

if w_coeff(4) == 1.,
  w    = w_coeff(1)+w_coeff(2)^(V^w_coeff(3)); % Wake coeff. : method 1
elseif w_coeff(4) == 2.,
  w    = w_coeff(1)/tanh(V*w_coeff(2));       % Wake coeff. : method 2
else
  disp(['ERROR : wrong definition of wake coefficient evaluation',...
    'method. Parameter w_method should be 1 or 2.']);
end;
V_A  = V*(1-w);                             % Advance velocity
V_kts= 1.94384*V;                           % Ship's velocity in knots
thdf = thdf_coeff(1)+thdf_coeff(2)...       % Thrust deduction factor
                *tanh(thdf_coeff(3)*(V_kts*(1852/3600)/sqrt(9.81*L_pp)))
                ;

FCT_N = @(x) (P_D/Nbr)/(2*pi*rho_w*D^5*x^3)-0.1*polyval(P_KQ10,V_A/(x*D));

if (P_D > 1e-5) || (V > 1e-5), % when P_D and V are not 0
    N_min = 1e-10;
    while FCT_N(N_min) < 0.,
        N_min = N_min*5;
    end;
    N_max = 1000;
    N_calc = fzero(FCT_N,[N_min N_max]);
    J_calc   = V_A/(N_calc*D);
    K_T_calc = polyval(P_KT, J_calc);
    K_Q_calc = 0.1*polyval(P_KQ10, J_calc);
    Q_calc   = K_Q_calc*rho_w*N_calc^2*D^5;     % Torque per propeller
    T_calc   = Nbr*K_T_calc*rho_w*N_calc^2*D^4; % Total thrust
    T_net_m0 = T_calc*(1-thdf);                 % Total net thrust w/o ice
        milling
    eta_D    = T_net_m0*V/P_D;                  % Propulsive efficiency
    T_net    = T_net_m0*(1-0.2*2*(H_milling/D)*tanh(V));
                                                % Total net thrust in ice milling
else
    N_calc   = 0;
    T_net    = 0;
end;
end
```

### *D.2.2 Thrust_calc_1prop.m*

```
function [T_net, N_calc] = Thrust_calc_1prop(P_D_1prop, V, fig)
%
% Calculation of the net thrust of one ship propeller, based on the
% propeller open-water characteristics.
%
% INPUTS :
% V                 [m/s]   Ship's velocity
% P_D_1prop         [W]     Delivered power to the considered propeller
% fig               [Bool]  Activate/Deactivate figures plotting
%
% OUTPUTS:
% T_net             [N]     Total net thrust
% N_calc            [1/s]   Revolution rate
%

global Ice_Water_data
global Ship_data
global Prop_data

% READING ICE WATER PARAMETERS
rho_w       = Ice_Water_data{5}(1); % Water density
H_milling   = Ice_Water_data{11};   % Propeller milling ice thickness

% READING SHIP PARAMETERS
L_pp        = Ship_data{2}(1);      % Ship's length

% READING INPUT PROPELLER DATA
Ref         = Prop_data{1};         % Propeller reference
D           = Prop_data{2};         % Propeller(s) diameter
w_coeff     = Prop_data{4};         % Wake coefficients
thdf_coeff  = Prop_data{5};         % Thrust deduction factor coefficients
J           = Prop_data{6};         % Advance number
KQ10        = Prop_data{7};         % Torque coefficient (x10)
KT          = Prop_data{8};         % Thrust coefficient
ETA0        = KT.*J./(KQ10/10*2*pi);% Propeller efficiency


% TRENDLINES CALCULATION
[P_KQ10,S_KQ10] = polyfit(J,KQ10,4);
[P_KT,S_KT]     = polyfit(J,KT,4);

R2_KQ10     = 1 - S_KQ10.normr^2 / norm(KQ10-mean(KQ10))^2;
R2_KT       = 1 - S_KT.normr^2 / norm(KT-mean(KT))^2;
if (R2_KQ10 < 0.95) || (R2_KT < 0.95),
    disp(['WARNING : Open-water characteristics poorly estimated. ',...
          'Please run Thrust_calc_1prop.m with fig=true to visualize ',...
          'the estimation.'])
end;

% REPRESENTATION OF PROPELLER KT-KQ-ETA0 VS J DIAGRAM
if fig,
    J_repr = 0:.01:(max(J)+.2);
    trend_KQ10 = polyval(P_KQ10, J_repr);
    trend_KT   = polyval(P_KT, J_repr);
    trend_ETA0 = interp1(J, ETA0, J_repr);

    figure('Position', [0 0 900 500]),
    plot(J, KQ10, 'o', 'Color', 'b', 'MarkerFaceColor', 'b');
    hold on,
    plot(J_repr, trend_KQ10, 'b');
    hold on,
    plot(J, KT, 's', 'Color', 'r', 'MarkerFaceColor', 'r');
    hold on,
    plot(J_repr, trend_KT, 'r');
    hold on,
    plot(J, ETA0, '^', 'Color', 'g', 'MarkerFaceColor', 'g');
    hold on,
    plot(J_repr, trend_ETA0, 'g');
    legend('10*KQ', '10*KQ (trend line)', 'KT', 'KT (trend line)',...
                                                'ETA0', 'ETA0 (trend line)');
    xlabel('J [-]', 'Fontsize', 12)
    ylabel('10*KQ, KT [-]', 'Fontsize', 12)
    title(['Propeller diagram - Reference : ', Ref], 'Fontsize', 12)
    axis([0 (max(J)+.2) 0 (max([max(KQ10) max(KT) max(ETA0)])+0.1)])
end;


% THRUST CALCULATION
if V < 1e-10,
    V = 1e-10; % To avoid stability problems
end;

if w_coeff(4) == 1.,
  w    = w_coeff(1)+w_coeff(2)^(V^w_coeff(3)); % Wake coeff. : method 1
elseif w_coeff(4) == 2.,
  w    = w_coeff(1)/tanh(V*w_coeff(2));       % Wake coeff. : method 2
else
  disp(['ERROR : wrong definition of wake coefficient evaluation',...
    'method. Parameter w_method should be 1 or 2.']);
end;
V_A  = V*(1-w);                             % Advance velocity
V_kts= 1.94384*V;                           % Ship's velocity in knots
thdf = thdf_coeff(1)+thdf_coeff(2)...       % Thrust deduction factor
                *tanh(thdf_coeff(3)*(V_kts*(1852/3600)/sqrt(9.81*L_pp)));

FCT_N = @(x) P_D_1prop/(2*pi*rho_w*D^5*x^3)-0.1*polyval(P_KQ10,V_A/(x*D));

if (P_D_1prop > 1e-5) || (V > 1e-5), % when P_D and V are not 0
    N_min = 1e-10;
    while FCT_N(N_min) < 0.,
        N_min = N_min*5;
    end;
    N_max = 1000;
    N_calc = fzero(FCT_N,[N_min N_max]);
    J_calc   = V_A/(N_calc*D);
    K_T_calc = polyval(P_KT, J_calc);
    K_Q_calc = 0.1*polyval(P_KQ10, J_calc);
    Q_calc   = K_Q_calc*rho_w*N_calc^2*D^5; % Torque per propeller
    T_calc   = K_T_calc*rho_w*N_calc^2*D^4; % Total thrust
    T_net_m0 = T_calc*(1-thdf);             % Total net thrust w/o ice milling
    eta_D    = T_net_m0*V/P_D_1prop;        % Propulsive efficiency
    T_net    = T_net_m0*(1-0.2*2*(H_milling/D)*tanh(V));
                                            % Total net thrust in ice milling
else
    N_calc   = 0;
    T_net    = 0;
end;

end
```

## D.3 Level Ice Resistance

### *D.3.1 Ice_resistance.m*

```
function [R_ice] = Ice_resistance(V, fig)
%
% Calculation of level ice resistance of an icebreaking ship, based on the
% Lindqvist theory modified by HSVA.
%
% INPUTS :
% V                 [m/s]   Ship's velocity
% fig               [Bool]  Activate/Deactivate figures plotting
%
% OUTPUT :
% R_ice             [N]     Total ice resistance
%

global Ice_Water_data
global Ship_data
global Other_data

% READING ICE WATER PARAMETERS
H_ice         = Ice_Water_data{2};     % Ice thickness
H_snow        = Ice_Water_data{3}(1);  % Snow thickness
H_snow_eff_H_snow = Ice_Water_data{3}(2);% Ratio H_snow_eff/H_snow
sigma_b_kPa   = Ice_Water_data{4};     % Ice bending strength (! in kPa)
rho_w         = Ice_Water_data{5}(1);  % Water density
rho_i         = Ice_Water_data{5}(2);  % Ice density
rho_s         = Ice_Water_data{5}(3);  % Snow density
E_kPa         = Ice_Water_data{6};     % Ice elastic modulus (! in kPa)
nu            = Ice_Water_data{7};     % Ice Poisson's ratio
k             = Ice_Water_data{8};     % Coeff. for breaking resistance
Lch_Lcusp     = Ice_Water_data{9};     % Charact. to cusp lengths ratio
Cov_B         = Ice_Water_data{10};    % Ship bottom ice coverage

% READING SHIP PARAMETERS
Ref           = Ship_data{1};     % Reference
L_pp          = Ship_data{2}(1); % Ship's length
B             = Ship_data{3};     % Ship's breadth
T             = Ship_data{4};     % Ship's draft
L_wedge       = Ship_data{5}(1); % Wedge_Length
alpha_wedge   = Ship_data{5}(2); % Max_Wedge_Angle
angles_0      = Ship_data{6};     % phi, alpha, psi on the centerline
angles_1_4    = Ship_data{7};     % phi, alpha, psi at 1/4 breadth
angles_2_4    = Ship_data{8};     % phi, alpha, psi at 2/4 breadth
angles_3_4    = Ship_data{9};     % phi, alpha, psi at 3/4 breadth
angles_15_16= Ship_data{10};    % phi, alpha, psi at 15/16 breadth
angles_mean   = Ship_data{11};    % phi, alpha, psi mean values
mu            = Ship_data{12};    % Friction coefficient
V_unit_ow     = Ship_data{15};    % Velocity unit for the OW resistance
                                  % empirical estimation
V_min_ow      = Ship_data{16};    % Minimum velocity for OW resistance
                                  % empirical estimation
coeff_ow  = Ship_data{17};    % Coefficients of the OW resistance
                                  % empirical estimation

% READING OTHER PARAMETERS
g             = Other_data{2};    % Gravitational constant

% INTERMEDIARY RESULTS
E               = E_kPa*1000;       % Elasticity modulus in SI units (N/m^2)
sigma_b       = sigma_b_kPa*1000; % Bending strength in SI units (N/m^2)

alpha_wedge_rad  = alpha_wedge*pi/180;
angles_0_rad     = angles_0*pi/180;

% Average angles on the breadth segments
angles_av_1_4_rad = 0.5*(angles_0+angles_1_4)*pi/180;
angles_av_2_4_rad = 0.5*(angles_1_4+angles_2_4)*pi/180;
angles_av_3_4_rad = 0.5*(angles_2_4+angles_3_4)*pi/180;
angles_av_4_4_rad = 0.5*(angles_3_4+angles_15_16)*pi/180;

angles_mean_rad  = angles_mean*pi/180;


L_ch    = (E*H_ice^3/(12*(1-nu^2)*rho_w*g))^.25;  % Ice charact. length
L_cusp = L_ch/Lch_Lcusp;                          % Cusp length
%n_cusp = B/(L_cusp*sin(angles_mean_rad(2)));     % Number of cusps
A_u     = max(0, B*(Cov_B*L_pp-T/tan(angles_mean_rad(1)) ...
                -0.25*B/tan(angles_mean_rad(2))));% Ice covered bottom area
%A_u_rel= A_u/(L_pp*B);                           % Relative bottom area
A_f     = B*T*sqrt((1/sin(angles_mean_rad(1)))^2 ...
            +(1/tan(angles_mean_rad(2)))^2);      % Bow area

H_snow_eff = H_snow_eff_H_snow*H_snow; % Effective snow thickness

% RESISTANCE CALCULATION
% Vertical force at the stem
F_v  = 0.5*sigma_b*H_ice^2;

% Crushing resistance
R_c  = F_v*(tan(angles_0_rad(1)) ...
      +mu*cos(angles_0_rad(1))/cos(angles_0_rad(3))) ...
       /(1-mu*sin(angles_0_rad(1))/cos(angles_0_rad(3)));

% Breaking resistance
R_b1 = k*sigma_b*(B/4)*(H_ice^3/L_cusp^2) ...
       *(tan(angles_av_1_4_rad(3))+mu*cos(angles_av_1_4_rad(1))) ...
       /(sin(angles_av_1_4_rad(2))*cos(angles_av_1_4_rad(3)))) ...
       *(1+1/cos(angles_av_1_4_rad(3)));
R_b2 = k*sigma_b*(B/4)*(H_ice^3/L_cusp^2) ...
       *(tan(angles_av_2_4_rad(3))+mu*cos(angles_av_2_4_rad(1))) ...
       /(sin(angles_av_2_4_rad(2))*cos(angles_av_2_4_rad(3)))) ...
       *(1+1/cos(angles_av_2_4_rad(3)));
R_b3 = k*sigma_b*(B/4)*(H_ice^3/L_cusp^2) ...
       *(tan(angles_av_3_4_rad(3))+mu*cos(angles_av_3_4_rad(1))) ...
       /(sin(angles_av_3_4_rad(2))*cos(angles_av_3_4_rad(3)))) ...
       *(1+1/cos(angles_av_3_4_rad(3)));
R_b4 = k*sigma_b*(B/4)*(H_ice^3/L_cusp^2) ...
       *(tan(angles_av_4_4_rad(3))+mu*cos(angles_av_4_4_rad(1))) ...
       /(sin(angles_av_4_4_rad(2))*cos(angles_av_4_4_rad(3)))) ...
       *(1+1/cos(angles_av_4_4_rad(3)));
R_b  = R_b1 + R_b2 + R_b3 + R_b4;

% Resistance due to loss of potential energy
R_p  = (H_ice*(rho_w-rho_i)+H_snow_eff*(rho_w-rho_s))*g*B*T*(B+T)/(B+2*T);

% Frictional resistance
R_f  = (H_ice*(rho_w-rho_i)+H_snow_eff*(rho_w-rho_s))*g*mu...
    *(A_u+A_f*cos(angles_mean_rad(1))*cos(angles_mean_rad(3)));

% Wedge resistance (! in [N/(m/s)^2])
if L_wedge == 0,
  r_w = 0;
else
  r_w = (2/3)*L_wedge*B*(H_ice*rho_i+H_snow*rho_s)...
                       *tan(alpha_wedge_rad)/L_wedge;
end;

% Open-water resistance
if strcmp(V_unit_ow, 'm/s'),
    V_ow = V;
elseif strcmp(V_unit_ow, 'kts'),
    V_ow = V*1.9438;
else
    disp(['Warning : unknown velocity unit for open-water resistance',...
        'calculation. [m/s] is considered.']);
    V_ow = V;
end;

if V >= V_min_ow, % Empirical relation in its range of validity
  R_ow = 1e3*polyval(coeff_ow, V_ow);
else       % Regression under the lower end of the validity range
  R_ow_min = 1e3*polyval(coeff_ow, V_min_ow);
  R_ow = R_ow_min*(V_ow^2)/(V_min_ow^2);
end;

% Total resistance
R_tot = (R_c+R_b)*(1+1.4*V/sqrt(g*H_ice))...
       +(R_p+R_f)*(1+9.4*V/sqrt(g*L_pp))...
       +r_w*V^2+(2/3)*R_ow;

if R_tot > R_ow,
  R_ice = R_tot;
else
  R_ice = R_ow;
end;

% REPRESENTATION OF OPEN-WATER RESISTANCE EMPIRICAL CURVE
if fig,
    V_repr = 0:.25:10;
    R_ow_repr = zeros(1,length(V_repr));
    for i = 1:length(V_repr),
        if V_repr(i) >= V_min_ow,% in the validity range of empirical rel.
            R_ow_repr(i) = 1e3*polyval(coeff_ow, V_repr(i));
        else % outisde validity range => regression
            R_ow_min = 1e3*polyval(coeff_ow, V_min_ow);
            R_ow_repr(i) = R_ow_min*(V_repr(i)^2)/(V_min_ow^2);
        end;
    end;

    figure('Position', [0 0 700 400]),
    plot(V_repr, R_ow_repr/1e3, 'b');
    if strcmp(V_unit_ow, 'm/s'),
        xlabel('$V$ $[m/s]$', 'Fontsize', 12, 'interpreter', 'latex')
    elseif strcmp(V_unit_ow, 'kts'),
        xlabel('$V$ $[kts]$', 'Fontsize', 12, 'interpreter', 'latex')
    else
        disp(['Warning : unknown velocity unit for open-water',...
        ' resistance calculation. [m/s] is considered.']);
        xlabel('$V$ $[m/s]$', 'Fontsize', 12, 'interpreter', 'latex')
    end;

    ylabel('\textit{Open-water resistance} $[kN]$', 'Fontsize', 12,...
           'interpreter', 'latex')
    title(['Open-water resistance curve - Reference : ', Ref],...
          'Fontsize', 12)
    axis([0 (max(V_repr)+1) 0 (1.1*max(R_ow_repr/1e3))])
end;

end
```

### *D.3.2 Discr_Ice_resistance.m*

```
function [R_c, R_b, R_p, R_f, r_w, R_ow] = Discr_Ice_resistance(V,...
          beta_el_rad)
%
% Calculation of ice resistance of an icebreaking ship, based on the
% Lindqvist theory modified by HSVA. As for Ice_resistance.m, this
% function considers a discretization of the bow in 8 elements, but here,
% a specific local drift angle can be given for earch of these elements.
% This function is used for the turning estimation of icebreaking ships.
%
% INPUTS :
% V                 [m/s]    Ship's velocity
% beta_el_rad       [rad]    Local drift angle at the elements
%
% OUTPUTS :
% R_c               [N]          Crushing resistance
% R_b               [N]          Bending resistance (on each element)
% R_p               [N]          Potential energy loss resistance
% R_f               [N]          Frictional resistance
% r_w               [N/(m/s)^2]  Wedge resistance
% R_ow              [N]          Open-water resistance
%

global Ice_Water_data
global Ship_data
global Other_data

% READING ICE WATER PARAMETERS
H_ice             = Ice_Water_data{2};     % Ice thickness
H_snow            = Ice_Water_data{3}(1);  % Snow thickness
H_snow_eff_H_snow = Ice_Water_data{3}(2);% Ratio H_snow_eff/H_snow
sigma_b_kPa       = Ice_Water_data{4};     % Ice bending strength (! in kPa)
rho_w             = Ice_Water_data{5}(1);  % Water density
rho_i             = Ice_Water_data{5}(2);  % Ice density
rho_s             = Ice_Water_data{5}(3);  % Snow density
E_kPa             = Ice_Water_data{6};     % Ice elastic modulus (! in kPa)
nu                = Ice_Water_data{7};     % Ice Poisson's ratio
k                 = Ice_Water_data{8};     % Coeff. for breaking resistance
Lch_Lcusp         = Ice_Water_data{9};     % Charact. to cusp lengths ratio
Cov_B             = Ice_Water_data{10};    % Ship bottom ice coverage

% READING SHIP PARAMETERS
L_pp          = Ship_data{2}(1); % Ship's length
B             = Ship_data{3};    % Ship's breadth
T             = Ship_data{4};    % Ship's draft
L_wedge       = Ship_data{5}(1); % Wedge_Length
alpha_wedge   = Ship_data{5}(2); % Max_Wedge_Angle
angles_0      = Ship_data{6};    % phi, alpha, psi on the centerline
angles_1_4    = Ship_data{7};    % phi, alpha, psi at 1/4 breadth
angles_2_4    = Ship_data{8};    % phi, alpha, psi at 2/4 breadth
angles_3_4    = Ship_data{9};    % phi, alpha, psi at 3/4 breadth
angles_15_16  = Ship_data{10};   % phi, alpha, psi at 15/16 breadth
angles_mean   = Ship_data{11};   % phi, alpha, psi mean values
mu            = Ship_data{12};   % Friction coefficient
V_unit_ow     = Ship_data{15};   % Velocity unit for the OW resistance
                                 % empirical estimation
V_min_ow      = Ship_data{16};   % Minimum velocity for OW resistance
                                 % empirical estimation
coeff_ow      = Ship_data{17};   % Coefficients of the OW resistance
                                 % empirical estimation

% READING OTHER PARAMETERS
g             = Other_data{2};

% INTERMEDIARY RESULTS
E             = E_kPa*1000;        %Elasticity modulus in SI units (N/m^2)
sigma_b       = sigma_b_kPa*1000;  %Bending strength in SI units (N/m^2)

alpha_wedge_rad  = alpha_wedge*pi/180;
angles_0_rad     = angles_0*pi/180;

% Average angles on the breadth segments
angles_av_1_4_rad = 0.5*(angles_0+angles_1_4)*pi/180;
angles_av_2_4_rad = 0.5*(angles_1_4+angles_2_4)*pi/180;
angles_av_3_4_rad = 0.5*(angles_2_4+angles_3_4)*pi/180;
angles_av_4_4_rad = 0.5*(angles_3_4+angles_15_16)*pi/180;

angles_mean_rad  = angles_mean*pi/180;


% hull angles in the moving direction

angles_m4_4_rad_d = [atan((tan(angles_av_4_4_rad(3)))...
                        *(sin(angles_av_4_4_rad(2)+beta_el_rad(1))));...
                     angles_av_4_4_rad(2)+beta_el_rad(1);...
                     angles_av_4_4_rad(3)];
angles_m3_4_rad_d = [atan((tan(angles_av_3_4_rad(3)))...
                        *(sin(angles_av_3_4_rad(2)+beta_el_rad(2))));...
                     angles_av_3_4_rad(2)+beta_el_rad(2);...
                     angles_av_3_4_rad(3)];
angles_m2_4_rad_d = [atan((tan(angles_av_2_4_rad(3)))...
                        *(sin(angles_av_2_4_rad(2)+beta_el_rad(3))));...
                     angles_av_2_4_rad(2)+beta_el_rad(3);...
                     angles_av_2_4_rad(3)];
angles_m1_4_rad_d = [atan((tan(angles_av_1_4_rad(3)))...
                        *(sin(angles_av_1_4_rad(2)+beta_el_rad(4))));...
                     angles_av_1_4_rad(2)+beta_el_rad(4);...
                     angles_av_1_4_rad(3)];

angles_p1_4_rad_d = [atan((tan(angles_av_1_4_rad(3)))...
                        *(sin(angles_av_1_4_rad(2)+beta_el_rad(5))));...
                     angles_av_1_4_rad(2)+beta_el_rad(5);...
                     angles_av_1_4_rad(3)];
angles_p2_4_rad_d = [atan((tan(angles_av_2_4_rad(3)))...
                        *(sin(angles_av_2_4_rad(2)+beta_el_rad(6))));...
                     angles_av_2_4_rad(2)+beta_el_rad(6);...
                     angles_av_2_4_rad(3)];
angles_p3_4_rad_d = [atan((tan(angles_av_3_4_rad(3)))...
                        *(sin(angles_av_3_4_rad(2)+beta_el_rad(7))));...
                     angles_av_3_4_rad(2)+beta_el_rad(7);...
                     angles_av_3_4_rad(3)];
angles_p4_4_rad_d = [atan((tan(angles_av_4_4_rad(3)))...
                        *(sin(angles_av_4_4_rad(2)+beta_el_rad(8))));...
                     angles_av_4_4_rad(2)+beta_el_rad(8);...
                     angles_av_4_4_rad(3)];


L_ch    = (E*H_ice^3/(12*(1-nu^2)*rho_w*g))^.25;  % Ice charact. length
L_cusp  = L_ch/Lch_Lcusp;                         % Cusp length
n_cusp  = B/(L_cusp*sin(angles_mean_rad(2)));     % Number of cusps
A_u     = max(0, B*(Cov_B*L_pp-T/tan(angles_mean_rad(1)))...
              -0.25*B/tan(angles_mean_rad(2))));% Ice covered bottom area
A_u_rel= A_u/(L_pp*B);                            % Relative bottom area
A_f     = B*T*sqrt((1/sin(angles_mean_rad(1)))^2 ...
           +(1/tan(angles_mean_rad(2)))^2);       % Bow area

H_snow_eff = H_snow_eff_H_snow*H_snow; % Effective snow thickness

% RESISTANCE CALCULATION
% Vertical force at the stem
F_v  = 0.5*sigma_b*H_ice^2;

% Crushing resistance
R_c  = F_v*(tan(angles_0_rad(1))...
       +mu*cos(angles_0_rad(1))/cos(angles_0_rad(3)))...
       /(1-mu*sin(angles_0_rad(1))/cos(angles_0_rad(3)));

% Breaking resistance
R_b_m4_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_m4_4_rad_d(3))+mu*cos(angles_m4_4_rad_d(1)))...
         /(sin(angles_m4_4_rad_d(2))*cos(angles_m4_4_rad_d(3))))...
         *(1+1/cos(angles_m4_4_rad_d(3)));
R_b_m3_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_m3_4_rad_d(3))+mu*cos(angles_m3_4_rad_d(1)))...
         /(sin(angles_m3_4_rad_d(2))*cos(angles_m3_4_rad_d(3))))...
         *(1+1/cos(angles_m3_4_rad_d(3)));
R_b_m2_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_m2_4_rad_d(3))+mu*cos(angles_m2_4_rad_d(1)))...
         /(sin(angles_m2_4_rad_d(2))*cos(angles_m2_4_rad_d(3))))...
         *(1+1/cos(angles_m2_4_rad_d(3)));
R_b_m1_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_m1_4_rad_d(3))+mu*cos(angles_m1_4_rad_d(1)))...
         /(sin(angles_m1_4_rad_d(2))*cos(angles_m1_4_rad_d(3))))...
         *(1+1/cos(angles_m1_4_rad_d(3)));
R_b_p1_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_p1_4_rad_d(3))+mu*cos(angles_p1_4_rad_d(1)))...
         /(sin(angles_p1_4_rad_d(2))*cos(angles_p1_4_rad_d(3))))...
         *(1+1/cos(angles_p1_4_rad_d(3)));
R_b_p2_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_p2_4_rad_d(3))+mu*cos(angles_p2_4_rad_d(1)))...
         /(sin(angles_p2_4_rad_d(2))*cos(angles_p2_4_rad_d(3))))...
         *(1+1/cos(angles_p2_4_rad_d(3)));
R_b_p3_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_p3_4_rad_d(3))+mu*cos(angles_p3_4_rad_d(1)))...
         /(sin(angles_p3_4_rad_d(2))*cos(angles_p3_4_rad_d(3))))...
         *(1+1/cos(angles_p3_4_rad_d(3)));
R_b_p4_4 = k*sigma_b*(B/8)*(H_ice^3/L_cusp^2)...
         *(tan(angles_p4_4_rad_d(3))+mu*cos(angles_p4_4_rad_d(1)))...
         /(sin(angles_p4_4_rad_d(2))*cos(angles_p4_4_rad_d(3))))...
         *(1+1/cos(angles_p4_4_rad_d(3)));


R_b  = [R_b_m4_4; R_b_m3_4; R_b_m2_4; R_b_m1_4;...
        R_b_p1_4; R_b_p2_4; R_b_p3_4; R_b_p4_4];

% Resistance due to loss of potential energy
R_p  = (H_ice*(rho_w-rho_i)+H_snow_eff*(rho_w-rho_s))*g*B*T*(B+T)/(B+2*T);

% Frictional resistance
R_f  = (H_ice*(rho_w-rho_i)+H_snow_eff*(rho_w-rho_s))*g*mu...
     *(A_u+A_f*cos(angles_mean_rad(1))*cos(angles_mean_rad(3)));

% Wedge resistance (! in [N/(m/s)^2])
if L_wedge == 0,
  r_w = 0;
else
  r_w = (2/3)*L_wedge*B*(H_ice*rho_i+H_snow*rho_s)...
                       *tan(alpha_wedge_rad)/L_wedge;
end;

% Open-water resistance
if strcmp(V_unit_ow, 'm/s'),
    V_ow = V;
elseif strcmp(V_unit_ow, 'kts'),
    V_ow = V*1.9438;
else
    disp(['Warning : unknown velocity unit for open-water resistance',...
          'calculation. [m/s] is considered.']);
    V_ow = V;
end;

if V >= V_min_ow, % Empirical relation in its range of validity
  R_ow = 1e3*polyval(coeff_ow, V_ow);
else        % Regression under the lower end of the validity range
  R_ow_min = 1e3*polyval(coeff_ow, V_min_ow);
  R_ow = R_ow_min*(V_ow^2)/(V_min_ow^2);
end;


R_tot = (R_c+sum(R_b))*(1+1.4*V/sqrt(g*H_ice))...
        +(R_p+R_f)*(1+9.4*V/sqrt(g*L_pp))...
        + r_w*V^2 ...
        +(2/3)*R_ow;

if R_tot < R_ow,
  R_c  = 0;
  R_b  = 0;
  R_f  = 0;
end;

end
```

### *D.3.3 Equil_level_ice.m*

```
function [V, T_net, N_calc] = Equil_level_ice(P_D, V_rel, disp_eq,...
                                               fig_local)
%
% Computes the equilibrium velocity and the associated propeller(s) thrust
% and revolution rate so that ship's total net thrust equals ice
% resistance, for a given power level.
%
% INPUTS :
% P_D             [W]     Total delivered power
% V_rel           [m/s]   Relative velocity to the ice (drift velocity,...)
% disp_eq         [Bool]  Activate/Deactivate equilibrium velocity display
% fig_local       [Bool]  Activate/Deactivate local figures plotting
%                         (Thrust and resistance diagrams)
%
% OUTPUTS:
% V               [m/s]   Ship's equilibrium velocity in level ice
% T_net           [N]     Total net thrust
% N_calc          [1/s]   Revolution rate
%

global Ship_data

P_D_max_MW = Ship_data{14}(1);% Ship's max delivered power (! in MW)

if P_D/(P_D_max_MW*1e6) < 1e-6,
    P_D = 1e-6*P_D_max_MW*1e6; % To avoid stability problems
end;

% Equilibrium function
FCT_V = @(V) Thrust_calc(P_D, V+V_rel, fig_local)...
           - Ice_resistance(V+V_rel, fig_local);

% Equilibrium calculation
try
    V     = fzero(FCT_V,[0.001 20]);
    % Verification
    [T_net, N_calc] = Thrust_calc(P_D, V+V_rel, fig_local);
    %R_ice = Ice_resistance(V+V_rel, Ice_Water_data, Ship_data,...
    %       Other_data, fig_local);
catch me
    if P_D_max_MW*1e6 <= 1e-6*P_D_max_MW*1e6,
        if disp_eq,
            disp('Zero initial power. Ship at rest.');
        end;
        V      = 0;
        T_net  = 0;
        %R_ice  = 0;
        N_calc = 0;
    elseif strcmp(me.identifier,...
            'MATLAB:fzero:ValuesAtEndPtsSameSign'),
        if disp_eq,
            disp(['Insufficient icebreaking power.',...
                ' Ship at rest.']);
        end;
        V      = 0;
        T_net  = 0;
        N_calc = 0;
    else
        disp('ERROR')
        disp(me);
        return
    end;
end;

if disp_eq,
    V_kts = V*1.94384449;

    disp('Level ice velocity :')
    disp(['V_lvl = ', num2str(V, 3),' m/s (', num2str(V_kts,3),' kts)'])
    disp(['N_lvl = ', num2str(N_calc),  ' rps' ])
    disp(['T_net_lvl = ', num2str(T_net1/1e3),  ' kN' ])
    disp(['R_ice_lvl = ', num2str(R_ice/1e3),  ' kN' ])
end;

end
```

## D.4 Resistance in Floes

### *D.4.1 Ice_resistance_floes.m*

```
function R_floes = Ice_resistance_floes(V, d_floe, C_ice)
%
% Calculation of the ice resistance of an icebreaking ship in floes.
%
% INPUTS :
% V                 [m/s]    Ship's velocity
% d_floe            [m]      Floe size
% C_ice             [-]      Ice concentration
%
% OUTPUT :
% R_floes           [N]      Ice resistance in floes
%

global Ice_Water_data
global Ship_data
global Other_data

% READING ICE WATER PARAMETERS
H_ice      = Ice_Water_data{2};      % Ice thickness


% READING SHIP PARAMETERS
L_pp         = Ship_data{2}(1); % Ship's length
L_bow        = Ship_data{2}(2); % Length of ship's bow
L_mid        = Ship_data{2}(3); % Length of ship's midbody
B            = Ship_data{3};    % Ship's breadth
T            = Ship_data{4};    % Ship's draft
angles_2_4   = Ship_data{8};    % phi, alpha, psi at 2/4 breadth
V_unit_ow    = Ship_data{15};   % Velocity unit for the OW resistance
                                % empirical estimation
V_min_ow     = Ship_data{16};   % Minimum velocity for OW resistance
                                % empirical estimation
coeff_ow     = Ship_data{17};   % Coefficients of the OW resistance
                                % empirical estimation

% READING OTHER PARAMETERS
g            = Other_data{2};   % Gravitational constant

%Floe size
d_floe_calc = max(min(d_floe, 10*L_pp),2);


A_wf = 1.2*B*L_bow; % Waterplane area of the bow


H_M  = H_ice;
H_F  = 0.26 + sqrt(H_M*B);
FN0  = 2.57/sqrt(L_pp*g);
FN   = V/(sqrt(L_pp*g));

check = max(min((L_pp*T/(B^2))^3,20),5);


%Angles
phi_rad   = angles_2_4(1)*pi/180;
alpha_rad = angles_2_4(2)*pi/180;
psi_deg   = angles_2_4(3);
psi_rad   = angles_2_4(3)*pi/180;

%Coefficients for channel resistance
C3 = 845;
C4 = 42;
C5 = 825;
C_mu = max(0.15*cos(phi_rad)+ sin(psi_rad)*sin(alpha_rad), 0.45);
if psi_deg <= 45,
    C_psi = 0;
else
    C_psi = 0.047*psi_deg-2.115;
end;


% Channel resistance
R_ch = C3*C_mu*((H_F + H_M)^2)*(B + C_psi*H_F)...
       + C4*L_mid*H_F^2 ...
       + C5*check*(A_wf/L_pp)*(FN/FN0)^2;


R_lvl = Ice_resistance(V, false);


% Open-water resistance
if strcmp(V_unit_ow, 'm/s'),
   V_ow = V;
elseif strcmp(V_unit_ow, 'kts'),
   V_ow = V*1.9438;
else
    disp(['Warning : unknown velocity unit for open-water resistance',...
        'calculation. [m/s] is considered.']);
    V_ow = V;
end;

if V >= V_min_ow, % Empirical relation in its range of validity
  R_ow = 1e3*polyval(coeff_ow, V_ow);
else        % Regression under the lower end of the validity range
  R_ow_min = 1e3*polyval(coeff_ow, V_min_ow);
  R_ow = R_ow_min*(V_ow^2)/(V_min_ow^2);
end;


% Resistance in floes
R_floes = max((((10*L_pp-d_floe_calc)/(10*L_pp-2))*R_ch...
          + (sqrt((d_floe_calc-2)/(10*L_pp)))*R_lvl)*C_ice^3,...
          R_ow);
```

### *D.4.2 Equil_floes.m*

```
function [V, T_net, N_calc] = Equil_floes(P_D, V_rel, d_floe, C_ice,...
                                disp_eq, fig_local)
%
% Computes the equilibrium velocity in floes and the associated
% propeller(s) thrust and revolution rate so that ship's total net thrust
% equals floe ice resistance, for a given power level.
%
% INPUTS :
% P_D               [W]      Total delivered power
% V_rel             [m/s]    Relative velocity to the ice (drift velocity,...)
% disp_eq           [Bool]   Activate/Deactivate equilibrium velocity display
% fig_local         [Bool]   Activate/Deactivate local figures plotting
%                            (Thrust and resistance diagrams)
%
% OUTPUTS:
% V                 [m/s]    Ship's equilibrium velocity in level ice
% T_net             [N]      Total net thrust
% N_calc            [1/s]    Revolution rate
%

global Ship_data

P_D_max_MW  = Ship_data{14}(1);% Ship's max delivered power (! in MW)

if P_D/(P_D_max_MW*1e6) < 1e-6,
    P_D = 1e-6*P_D_max_MW*1e6; % To avoid stability problems
end;

% Equilibrium function
FCT_V = @(V) Thrust_calc(P_D, V+V_rel, fig_local)...
             - Ice_resistance_floes(V+V_rel, d_floe, C_ice);

% Equilibrium calculation
try
    V     = fzero(FCT_V,[0.001 20]);
    % Verification
    [T_net, N_calc] = Thrust_calc(P_D, V+V_rel, fig_local);
    %R_fl = Ice_resistance_floes(V+V_rel, d_floe, C_ice, fig_local);
catch me
    if P_D_max_MW*1e6 <= 1e-6*P_D_max_MW*1e6,
        if disp_eq,
            disp('Zero initial power. Ship at rest.');
        end;
        V      = 0;
        T_net  = 0;
        %R_fl  = 0;
        N_calc = 0;
    elseif strcmp(me.identifier,...
            'MATLAB:fzero:ValuesAtEndPtsSameSign'),
        if disp_eq,
            disp(['Insufficient icebreaking power.',...
                ' Ship at rest.']);
        end;
        V      = 0;
        T_net  = 0;
        N_calc = 0;
    else
        disp('ERROR')
        disp(me);
        return
    end;
end;

if disp_eq,
    V_kts = V*1.94384449;

    disp('Equilibrium velocity in floes:')
    disp(['V_fl = ', num2str(V, 3),' m/s (', num2str(V_kts,3),' kts)'])
    disp(['N_fl = ', num2str(N_calc), ' rps' ])
    disp(['T_net_fl = ', num2str(T_net/1e3), ' kN' ])
    disp(['R_floes = ', num2str(R_ice/1e3), ' kN' ])
end;

end
```

## D.5 Moving through Ice Ridges

### *D.5.1 Ice_ridge.m*

```
function [time, x, x_dot, x_ddot, R_ice, T_net, P_D, N_calc]=Ice_ridge(...
                    P_D_0, time_init, time_end, fig_local, fig_main)
%
% Calculation of an icebreaker motion through an ice ridge.
%
% INPUTS :
% P_D_0             [W]     Initial ship's delivered power
% time_init         [s]     Time in initial conditions before power increase
% time_end          [s]     Time in initial conditions after ridge breaking
% fig_local         [Bool] Activate/Deactivate local figures plotting
%                           (Thrust and resistance diagrams)
% fig_main          [Bool] Activate/Deactivate main figures plotting
%                           (Results of the ridge breaking process)
%
% OUTPUTS :
% time              [s]     Time series of the results
% x                 [m]     Position vector
% x_dot             [m]     Velocity vector
% x_ddot            [m]     Acceleration vector
% R_ice             [N]     Total ice resistance vector
% T_net             [N]     Total net thrust vector
% P_D               [W]     Delivered power vector
% N_calc            [1/s]   Propeller(s) revolution rate vector
%

global Ice_Water_data
global Ship_data
global Ridge_data
global Other_data

% Ship parameters
Ref                = Ship_data{1};    % Reference
L_bow              = Ship_data{2}(2); % Length of ship's bow
L_mid              = Ship_data{2}(3); % Length of ship's midbody
B                  = Ship_data{3};    % Ship's breadth
mu                 = Ship_data{12};   % Friction coefficient
m_ship_ton         = Ship_data{13}(1);% Ship displacement (! tons)
m_add_coeff        = Ship_data{13}(2);% Added mass coefficient m_add/m_ship
P_D_max_MW         = Ship_data{14}(1);% Ship's max delivered power (! MW)
time_to_P_D_max    = Ship_data{14}(2);% Time to power up from 0 to P_D_max
time_P_D_max_to_0  = Ship_data{14}(3);% Time to power down from P_D_max

% Ice water parameters
rho_w          = Ice_Water_data{5}(1);  % Water density
rho_i          = Ice_Water_data{5}(2);  % Ice density

% Ridge parameters
L_ridge        = Ridge_data{2}; % Ridge length
H_K            = Ridge_data{3}; % Keel depth of the ridge
L_adv_CD       = Ridge_data{4}; % Distance travelled at full speed
                                % (V > V_max_95) before the ridge
L_adv_HI       = Ridge_data{5}; % Distance travelled at full power after
                                % the ridge
k_ridge        = Ridge_data{6}; % Factor k for the ridge maximum resistance
N_ridge        = Ridge_data{7}; % Factor N for the ridge maximum resistance
                                % !!! N and k vary with ship and ramming
phi_s_deg      = Ridge_data{8}; % Inner friction angle for Yield resistance
                                % !!! Large range of values !!!
eta_ice_ridge  = Ridge_data{9}; % Ice porosity in the ridge

% Other parameters
g              = Other_data{2};  % Gravitational constant
dt             = Other_data{3};  % Time step


%%%%% CALCULATION OF INITIAL VELOCITY IN LEVEL ICE v_0 %%%%%
if P_D_0/(P_D_max_MW*1e6) < 1e-6,
    P_D_0 = 1e-6*P_D_max_MW*1e6; % To avoid stability problems
end;

% Equilibrium function
FCT_V = @(V) Thrust_calc(P_D_0, V, fig_local)...
             - Ice_resistance(V, fig_local);

% Equilibrium calculation
try
    v_0     = fzero(FCT_V,[0.001 20]);
    % Verification
    [T_net_0, N_calc_0] = Thrust_calc(P_D_0, v_0, fig_local);
    R_ice_0 = Ice_resistance(v_0, fig_local);
catch me
    if P_D_0 <= 1e-6*P_D_max_MW*1e6,
            disp('Zero initial power. Ship initially at rest.');
        v_0 = 0;
        T_net_0  = 0;
        R_ice_0  = 0;
        N_calc_0 = 0;
    elseif strcmp(me.identifier,'MATLAB:fzero:ValuesAtEndPtsSameSign'),
        disp(['Insufficient initial icebreaking power.',...
            ' Ship initially at rest.']);
        v_0 = 0;
        T_net_0 = 0;
        R_ice_0 = 0;
        N_calc_0 = 0;
    else
        disp('ERROR')
        disp(me);
        return
    end;
end;
v_0_kts = v_0*1.94384449;

disp('Initial velocity :')
disp(['v_0 = ', num2str(v_0, 3),' m/s (', num2str(v_0_kts,3),' kts)'])
disp(['N_0 = ', num2str(N_calc_0),  ' rps' ])


%%%%% CALCULATION OF V_max_th (maximum velocity in level ice) %%%%%
P_D_max = P_D_max_MW*1e6;
FCT_V_max_th = @(V_max_th) Thrust_calc(P_D_max, V_max_th, fig_local)...
               - Ice_resistance(V_max_th, fig_local);
try
    V_max_th = fzero(FCT_V_max_th,[0.0001 20]);
catch me
    if strcmp(me.identifier,'MATLAB:fzero:ValuesAtEndPtsSameSign'),
        disp(['ERROR : Unable to compute an equilibrium velocity. ',...
              'Insufficient Icebreaker power.']);
        return
    else
        disp('ERROR')
        disp(me);
        return
    end;
end;
V_max_th_kts = V_max_th*1.94384449;

% Verification
[T_net_max_th, N_calc_max] = Thrust_calc(P_D_max, V_max_th, fig_local);
R_ice_max_th = Ice_resistance(V_max_th, fig_local);

disp('Theoretical maximum velocity in level ice:')
disp(['V_max_th = ', num2str(V_max_th, 3),' m/s (',...
                     num2str(V_max_th_kts,3),' kts)'])
disp(['N_0 = ', num2str(N_calc_max),  ' rps' ])


%%%%% SIMULATION OF 0-A-B-C-D-E-F-G-H-I-J-K-End %%%%%

V_max_95 = 0.95*V_max_th;

% Initialisation : 1st guess (exact size is unknown)
lgth_1   = 1000;            % Base length for initialisation
x        = zeros(1,lgth_1);% [m]     initialisation of position vector
x_dot    = zeros(1,lgth_1);% [m/s]   initialisation of velocity vector
x_ddot   = zeros(1,lgth_1);% [m/s^2] initialisation of acceleration vector

P_D      = zeros(1,lgth_1);
T_net    = zeros(1,lgth_1);
N_calc   = zeros(1,lgth_1);
R_ice  = zeros(1,lgth_1);

x(1)     = 0;
x_dot(1) = v_0;
x_ddot(1)= 0;
P_D(1)   = P_D_0;
T_net(1) = T_net_0;
N_calc(1)= N_calc_0;
R_ice(1) = R_ice_0;

disp('Start');
phase = '0A';
disp(['phase ', phase]);

Ridge_OK      = false;
Ridge_ramming = false;
k = 2;
time_0 = 0;
while not(Ridge_OK) && not(Ridge_ramming),
    % Check if vectors size has to be increased
    if mod(k, lgth_1)==0,
        x        = horzcat(x,      zeros(1,lgth_1));
        x_dot    = horzcat(x_dot,  zeros(1,lgth_1));
        x_ddot   = horzcat(x_ddot,zeros(1,lgth_1));
        P_D      = horzcat(P_D,    zeros(1,lgth_1));
        T_net    = horzcat(T_net,  zeros(1,lgth_1));
        N_calc   = horzcat(N_calc,zeros(1,lgth_1));
        R_ice  = horzcat(R_ice,  zeros(1,lgth_1));
    end;

  % Ship's delivered power
    if strcmp(phase, '0A'),
        P_D(k) = P_D_0;
    elseif strcmp(phase, 'AB'),
        % Linear increase of the delivered power up to P_D_max
        P_D(k) = P_D_0 + ((k-1)*dt-time_A)*P_D_max/time_to_P_D_max;
    elseif    strcmp(phase,'BC') || strcmp(phase,'CD')...
           || strcmp(phase,'DE') || strcmp(phase,'EF')...
           || strcmp(phase,'FG') || strcmp(phase,'GH')...
           || strcmp(phase,'HI'),
        % Maximum delivered power
        P_D(k) = P_D_max;
    elseif strcmp(phase, 'IJ'),
        % Linear decrease of the delivered power up to P_D_0
        P_D(k) = P_D_max...
            - ((k-1)*dt-time_I)*P_D_max/time_to_P_D_max;
    elseif (strcmp(phase, 'JK') || strcmp(phase, 'KEnd')),
        % Back to initial delivered power P_D_0
        P_D(k) = P_D_0;
    end;

  % Thrust calculation
   [T_net(k), N_calc(k)] = Thrust_calc(P_D(k), x_dot(k-1), fig_local);


  % Resistance calculation
   if  strcmp(phase, 'DE'),
     R_ice(k) = R_ice_D + (x(k-1)-x_D)*(R_ice_E-R_ice_D)/L_DE;
     elseif strcmp(phase, 'EF'),
     R_ice(k) = A_exp*exp(-B_exp*x(k-1));
     elseif strcmp(phase, 'FG'),
     R_ice(k) = R_ice_mid;
     elseif strcmp(phase, 'GH'),
     R_ice(k) = real(A_tanh + B_tanh*tanh(-m_tanh*(x(k-1)-C_tanh)));
   else
     R_ice(k) = Ice_resistance(x_dot(k-1), fig_local);
     end


  % Acceleration, velocity and position
     if (strcmp(phase,'0A'))||strcmp(phase,'AB')||strcmp(phase,'KEnd'))...
        &&(T_net(k)-R_ice(k)<=0.),
         x_ddot(k) = 0;
         x_dot(k)  = v_0;
         x(k)      = x(k-1);
     else
         x_ddot(k) = (1/((1+m_add_coeff)*1e3*m_ship_ton))...
                     *(T_net(k)-R_ice(k));
         x_dot(k)  = x_dot(k-1) + x_ddot(k)*dt;
         x(k)      = x(k-1) + x_dot(k)*dt;
     end;
    %x_ddot(k),x_dot(k),x(k),

     % Steps
     % Phase 0A : Ship advances at initial speed
     % Phase AB : Increase of power until P_D_max
     if strcmp(phase,'0A') && ((k-1)*dt>=time_init),
         phase = 'AB';
         disp(['phase ', phase]);
         time_A = (k-1)*dt;
     % Phase BC : Acceleration until V_max_95 = 95% of max V in level ice
     elseif strcmp(phase,'AB') &&...
             (((k-1)*dt-time_A)>=time_to_P_D_max*(1-P_D_0/P_D_max)),
         phase = 'BC';
         disp(['phase ', phase]);
         time_B = (k-1)*dt;
```

```
    % Phase CD : Ship advances of L_adv_CD before reaching the ridge
    elseif strcmp(phase,'BC') && (x_dot(k) > V_max_95),
        phase = 'CD';
        disp(['phase ', phase]);
        x_C = x(k);
        time_C = (k-1)*dt;

    % Phase DE : Ships moves through the ridge, until fore shoulder
    % reaches the ridge centreline
    elseif strcmp(phase,'CD') && (x(k)-x_C > L_adv_CD),
        phase = 'DE';
        disp(['phase ', phase]);

        R_ice_D = R_ice(k);
        x_D     = x(k);
        time_D = (k-1)*dt;


        % Yield resistance
        phi_s_rad = phi_s_deg*pi/180;
        R_yield   = 0.5*((1+sin(phi_s_rad))/(1-sin(phi_s_rad)))...
            *(1-eta_ice_ridge)*rho_i*g*(1-rho_i/rho_w)*H_K^2;

        % Calculation of R_ice_E = R_ice_max (max resistance in the ridge)
        v_pen      = x_dot(k);
        R_ice_E = (k_ridge*v_pen/sqrt(g*H_K))*(B*R_yield +...
            2*mu*L_bow*R_yield) + 0.63*(k_ridge*v_pen/sqrt(g*H_K))...
            *(2*mu*R_yield*N_ridge*L_mid);

        % Length of phase D-E
        L_DE = L_bow + L_ridge/2;

    % Phase EF : Ship's bow is leaving the ridge
    elseif strcmp(phase, 'DE') && (x(k)-x_D > L_DE),
        phase = 'EF';
        disp(['phase ', phase]);

        R_ice_E = R_ice(k);
        x_E     = x(k);
        time_E  = (k-1)*dt;

        % Length of phase E-F
        L_EF = L_ridge/2;
        x_F  = x_E + L_EF;

        % Midbody resistance (cannot be less than level ice resistance)
        R_ice_mid = 0.63*R_ice_E;
        if R_ice_mid < R_ice_max_th,
            R_ice_mid = R_ice_max_th;
            disp(['Parallel midbody resistance too low.',...
                  ' Level ice resistance is considered.']);
        end;

        % Exponential decay of R_ice until R_mid : R_ice = A*exp(-B*x)
        A_exp = R_ice_E*(R_ice_mid/R_ice_E)^(x_E/(x_E-x_F));
        B_exp = (1/(x_E-x_F))*log(R_ice_mid/R_ice_E);

    % Phase FG : Only the ship's midbody is in contact with the ridge
    elseif strcmp(phase, 'EF') && (x(k)-x_E > L_EF),
        phase = 'FG';
        disp(['phase ', phase]);

        time_F = (k-1)*dt;
        new_it_H = false; % Indicates 1st iteration for H (see below)

        % Length of phase F-G
        L_FG = L_mid-L_ridge;  % if L_ridge < L_midbody (> is unexpected)

    % Phase GH : Midbody is leaving the ridge
    elseif strcmp(phase, 'FG') && (x(k)-x_F > L_FG),
        phase = 'GH';
        if ~new_it_H,
            disp(['phase ', phase]);
        else
            disp(['phase ', phase, ' (new iteration)']);
        end;
        R_ice_G = R_ice_mid;
        x_G     = x(k);
        time_G  = (k-1)*dt;
        k_G     = k;

        % Length of phase GH
        L_GH    = L_ridge;
        if not(exist('R_ice_H_th', 'var')),
            R_ice_H_th = R_ice_max_th;
            N_it = 1;
        end;
        x_H   = x_G + L_GH;

        % Resistance decreases smoothly until R_ice_max_th :
        % R_ice = A+B*tanh(-m*(x-C))
        A_tanh = (R_ice_G + R_ice_H_th)/2;
        B_tanh = (R_ice_G - R_ice_H_th)/2;
        C_tanh = (x_G + x_H)/2;
        m_tanh = (-1/(x_H-C_tanh))*atanh((1.01*R_ice_H_th-A_tanh)/B_tanh);

    % Phase HI : Ship is in level ice, keeping full power to recover speed
    elseif strcmp(phase, 'GH') && (x(k)-x_G > L_GH),
        R_ice_H = Ice_resistance(x_dot(k), fig_local);

        if abs(R_ice_H - R_ice(k))/R_ice_H > 1e-2,
            % we have to iterate to get R_ice_H
            new_it_H = true;
            if N_it == 1,
                f0_n=abs(R_ice_H - R_ice(k))/R_ice_H;
                x0_n=R_ice_max_th;
                R_ice_H_th = R_ice_H;
            else
                f0_nm1=f0_n;
                x0_nm1=x0_n;
                f0_n=abs(R_ice_H - R_ice(k))/R_ice_H;
                x0_n=R_ice_H_th;
                deriv = (f0_n-f0_nm1)/(x0_n-x0_nm1);
                R_ice_H_th = R_ice_H_th - f0_n/deriv;
            end;

            phase = 'FG';
            k = k_G;
            N_it = N_it+1;
        else
            phase = 'HI';
            disp(['phase ', phase]);
            x_H     = x(k);
            time_H  = (k-1)*dt;
        end;

    % Phase IJ : Decrease of power until P_D_0
    elseif strcmp(phase, 'HI') && (x(k)-x_H > L_adv_HI),
        phase = 'IJ';
        disp(['phase ', phase]);
        time_I  = (k-1)*dt;

    % Phase JK : Power at initial level, velocity is decreasing until v_0
    elseif strcmp(phase, 'IJ') && ...
            ((k-1)*dt-time_I > time_P_D_max_to_0*(1-P_D_0/P_D_max)),
        phase = 'JK';
        disp(['phase ', phase]);
        time_J = (k-1)*dt;

    % Phase KEnd : Ship has recovered its initial conditions in level ice
    elseif strcmp(phase, 'JK') && ...
            (((x_dot(k) > 0.95*v_0) && (x_dot(k) < 1.05*v_0)) ||...
            (x_dot(k) < 0.001)),
        % last part of the logical condition is for the case v_0 = 0.
        phase = 'KEnd';
        disp(['phase ', phase]);
        time_K = (k-1)*dt;

    % Phase End : simulation during time_end with initial conditions
    elseif strcmp(phase, 'KEnd') && (((k-1)*dt-time_K) >= time_end),
        disp('End');
        time_End = (k-1)*dt;
        Ridge_OK = true;
    end;

  % Check if ramming is required
    if (strcmp(phase,'DE') || strcmp(phase,'EF') || strcmp(phase,'FG') ...
            || strcmp(phase, 'GH')) && (x_dot(k) < 0.001),
        disp(['Unable to break ridge with the first advance.',...
              'Ramming is required.'])
        disp('Simulation stopped.')
        Ridge_ramming = true;
    end;


  k = k+1;
end;

if not(Ridge_ramming)
    time_steps = [time_0 time_A time_B time_C time_D time_E time_F ...
                  time_G time_H time_I time_J time_K time_End];
end;

% Suppression of the unused vector lengths
x      ((end-(lgth_1-mod(k, lgth_1))):end) = [];
x_dot  ((end-(lgth_1-mod(k, lgth_1))):end) = [];
x_ddot ((end-(lgth_1-mod(k, lgth_1))):end) = [];
P_D    ((end-(lgth_1-mod(k, lgth_1))):end) = [];
T_net  ((end-(lgth_1-mod(k, lgth_1))):end) = [];
N_calc ((end-(lgth_1-mod(k, lgth_1))):end) = [];
R_ice  ((end-(lgth_1-mod(k, lgth_1))):end) = [];

% Time vector
time = 0:dt:(dt*length(x)-dt);

% PRESENTATION OF THE RESULTS

if fig_main,

    figure('Position', [0 0 900 700]),

    subplot(3,1,1)
    plot(time, x, 'b')
    hold on,
    plot(time, 100*x_dot, 'r')
    hold on,
    plot(time, 1000*x_ddot, 'g')
    hold on,
    plot(time_steps, 1.1*min([x 100*x_dot 1000*x_ddot])...
         *ones(1,length(time_steps)),'^', 'Color', 'b',...
         'MarkerFaceColor', 'b', 'MarkerSize', 7);
    h_leg =legend('Displacement [m]',...
                  'Velocity [m/s]',...
                  'Acceleration [m/s^2]',...
                  'Location', 'NorthWest', 'Orientation', 'Horizontal');
    set(h_leg, 'FontSize', 9)
    min_yplot1 = 1.1*min([x 100*x_dot 1000*x_ddot]);
    max_yplot1 = 1.5*max([x 100*x_dot 1000*x_ddot]);
    hold on,
    stem(time_steps, max_yplot1*ones(length(time_steps),1),'k:',...
         'marker','none', 'BaseValue', min_yplot1, 'LineWidth', 0.5)
    xlabel('$time$ $[s]$', 'Fontsize', 12, 'Interpreter', 'latex')
    ylabel('$x$, $100\dot{x}$, $1000\ddot{x}$',...
           'Fontsize', 12, 'Interpreter', 'latex')
    title(['Simulation results - Reference : ', Ref], 'Fontsize', 12)
    axis([0 time(end) min_yplot1 max_yplot1])

    subplot(3,1,2)
    plot(time, T_net/1e3, 'b')
    hold on,
    plot(time, R_ice/1e3, 'r')
    h_leg = legend('T_{net}', 'R_{ice}', 'Location', 'NorthEast');
    set(h_leg, 'FontSize', 9)
    max_yplot2 = 1.1*max([T_net R_ice]/1e3);
    hold on,
    stem(time_steps, max_yplot2*ones(length(time_steps),1),'k:',...
         'marker','none', 'BaseValue', 0, 'LineWidth', 0.5)
    xlabel('$time$ $[s]$', 'Fontsize', 12, 'Interpreter', 'latex')
    ylabel('$T_{net}$, $R_{ice}$ $[kN]$',...
           'Fontsize', 12, 'Interpreter', 'latex')
    axis([0 time(end) 0 max_yplot2])

    subplot(3,1,3)
    plot(time, P_D/1e6, 'b')
    hold on,
    plot(time, N_calc, 'r')
    h_leg = legend('Delivered power', 'Propeller rate', 'Location',...
                   'NorthEast');
    set(h_leg, 'FontSize', 9)
    max_yplot3 = 1.4*max([P_D/1e6 N_calc]);
    hold on,
    stem(time_steps, max_yplot3*ones(length(time_steps),1),'k:',...
         'marker','none', 'BaseValue', 0, 'LineWidth', 0.5)
    xlabel('$time$ $[s]$', 'Fontsize', 12, 'Interpreter', 'latex')
    ylabel('$P_{D} [MW]$, $N [rps]$','Fontsize',12,'Interpreter','latex')
    axis([0 time(end) 0 max_yplot3])

end;

end
```

## D.6 Manoeuvrability Estimation

### *D.6.1 Calc_circle_V.m*

```
function [P_D, beta_prop_deg, beta_drift_deg] = Calc_circle_V( ...
                            R_turning, V_turning, turn_sign, prop_mode,...
                            turn_floes, fig_local)
%
% Calculation of turning circle. From a given turning configuration
% (R_turning and V_turning), this function evaluates the drift angle,
% the required delivered power and the propeller(s) angle.
%
% INPUTS :
% R_turning           [m]       Turning radius
% V_turning           [m/s]     Velocity of the ship's centre of gravity
% turn_sign           [-]       -1 for turn to starboard, +1 for turn to port
% prop_mode           [-]       Pod(s) rotating mode :
%                                 1: All pods are rotating, or rudder mode
%                                 2: 2 pods, only the pod outside turn rotates
%                                 3: 2 pods, only the pod inside turn rotates
% turn_floes          [-]       0 for turn in level ice, 1 for turn in floes
% fig_local           [Bool]    Activate/Deactivate figures plotting for
%                               hull waterline estimation
% OUTPUTS:
% P_D                 [W]       Ship's delivered power
% beta_prop_deg       [deg]     Pod(s)/rudder angle
% beta_drift_deg      [deg]     Ship's drift angle
%

global Ship_data
global Prop_data

% Reading input propeller data
Nbr_prop = Prop_data{3};      % Number of propellers
y_prop   = Prop_data{9}(2,:);% y-coord. of propellers location from aft

% Reading ship parameters
P_D_max_MW = Ship_data{14}(1); % Ship's max delivered power (! in MW)

% Vector of rotating propellers (pods)
%   1 : pod rotates in order to turn
%   0 : pod is used for thrust forward only
switch prop_mode,
    case 1, % All propellers are rotating
        Prop_vect = ones(1, Nbr_prop);
    case 2, % 2 pods, only the pod outside the turn rotates
        if turn_sign == -1,
        % turn to starboard
            [y,k_y] = min(y_prop);
        else
        % turn to port
            [y,k_y] = max(y_prop);
        end;
        Prop_vect = zeros(1, Nbr_prop);
        Prop_vect(k_y) = 1;
    case 3, % 2 pods, only the pod inside the turn rotates
        if turn_sign == -1,
        % turn to starboard
            [y,k_y] = max(y_prop);
        else
        % turn to port
            [y,k_y] = min(y_prop);
        end;
        Prop_vect = zeros(1, Nbr_prop);
        Prop_vect(k_y) = 1;
end;

% Maximum velocity in linear motion
FCT_P_D_max   = @(V_P_D_max) Thrust_calc(P_D_max_MW*1e6, V_P_D_max,...
                            false) - Ice_resistance(V_P_D_max, false);
V_P_D_max   = fzero(FCT_P_D_max,[0 100]);

if V_turning > 2*V_P_D_max,
    disp(['ERROR : Required velocity (', num2str(V_turning,3),' m/s) ',...
          'is more than twice the maximum achievable velocity (',...
          num2str(V_P_D_max,3),' m/s). Please increase maximum ice ',...
          'breaking power or review input parameters.'])
    error('SIMULATION ABORTED')
end;

% Initial point for delivered power : Linear motion data
FCT_lin   = @(P_D) Thrust_calc(P_D, V_turning, false)...
                 - Ice_resistance(V_turning, false);
P_D_lin   = fzero(FCT_lin,[1e3 2000e6]);


% System definition
FCT_turning = @(calc_turning) Turning_circle(V_turning, R_turning,...
                            calc_turning(1),calc_turning(2)*Prop_vect,...
                            calc_turning(3), turn_sign, turn_floes,...
                            fig_local);

options = optimset('MaxFunEvals', 1000, 'MaxIter', 200, 'Display', 'off');


% Calculation for a grid of initial points
P_D_calc            = zeros(3,3);
beta_prop_deg_calc  = zeros(3,3);
beta_drift_deg_calc = zeros(3,3);
flag_calc           = zeros(3,3);

coeff_P_D_init = [1 5 10];
beta_prop_init = [1 25 75];

for i=1:3,
    for j=1:3,
        % disp([num2str(3*(i-1)+j),'/9']) % Counter
        % Initial values
        calc_turning_0 = [ coeff_P_D_init(j)*P_D_lin,...
                           -turn_sign*beta_prop_init(i),...
                           -turn_sign*5];
        % Resolution
        [rslt_turning,fval,exitflag,output] = fsolve(FCT_turning,...
                                                calc_turning_0, options);

        P_D_calc(i,j)           = rslt_turning(1);%Required deliv. power
        beta_prop_deg_calc(i,j) = rslt_turning(2);%Pod(s) angle
        beta_drift_deg_calc(i,j)= rslt_turning(3);%Drift angle
        flag_calc(i,j)          = exitflag;
    end;
end;

%Analysis of the simulation
P_D_r            = reshape(P_D_calc, 1,9);
beta_prop_deg_r  = reshape(beta_prop_deg_calc, 1,9);
beta_drift_deg_r = reshape(beta_drift_deg_calc, 1,9);
flag_r           = reshape(flag_calc, 1,9);

l=1;
while l<=length(P_D_r),
    if (flag_r(l) <= 0)                 ||...
       (abs(beta_prop_deg_r(l)) > 90)   ||...
       (P_D_r(l) < 0.99*P_D_lin),
        P_D_r(l)          = [];
        beta_prop_deg_r(l)  = [];
        beta_drift_deg_r(l) = [];
        flag_r(l)           = [];
    else
    l=l+1;
    end;
end;
[min_P_D_diff, l_min_P_D] = min(P_D_r-P_D_lin);

if ~isempty(P_D_r),
    P_D            = P_D_r(l_min_P_D);          %Required delivered power
    beta_prop_deg  = beta_prop_deg_r(l_min_P_D); %Pod(s) angle
    beta_drift_deg = beta_drift_deg_r(l_min_P_D);%Drift angle
else
    % Unable to find a turning velocity for the requested power level
    P_D            = NaN; %Required delivered power
    beta_prop_deg  = NaN; %Pod(s) angle
    beta_drift_deg = NaN; %Drift angle
end;

end
```

### *D.6.2 Calc_circle_P_D.m*

```
function [V_turning, beta_prop_deg, beta_drift_deg] = Calc_circle_P_D(...
                    R_turning, P_D, turn_sign, prop_mode, turn_floes,...
                    fig_local)
%
% Calculation of turning circle at given turning radius and power level.
% From these two parameters, this function evaluates the maximum turning
% velocity.
%
% INPUTS :
% R_turning           [m]       Turning radius
% P_D                 [W]       Ship's delivered power
% turn_sign           [-]       -1 for turn to starboard, +1 for turn to port
% prop_mode           [-]       Pod(s) rotating mode :
%                                 1: All pods are rotating, or rudder mode
%                                 2: 2 pods, only the pod outside turn rotates
%                                 3: 2 pods, only the pod inside turn rotates
% turn_floes          [-]       0 for turn in level ice, 1 for turn in floes
% fig_local           [Bool]    Activate/Deactivate figures plotting for
%                               hull waterline estimation
% OUTPUTS:
% V_turning           [m/s]     Velocity of the ship's centre of gravity
% beta_prop_deg       [deg]     Pod(s)/rudder angle
% beta_drift_deg      [deg]     Ship's drift angle
%

global Prop_data

% Reading input propeller data
Nbr_prop = Prop_data{3};      % Number of propellers
y_prop   = Prop_data{9}(2,:);% y-coord. of propellers location from aft

% Vector of rotating propellers (pods)
%   1 : pod rotates in order to turn
%   0 : pod is used for thrust forward only
switch prop_mode,
    case 1, % All propellers are rotating
        Prop_vect = ones(1, Nbr_prop);
    case 2, % 2 pods, only the pod outside the turn rotates
        if turn_sign == -1,
        % turn to starboard
            [y,k_y] = min(y_prop);
        else
        % turn to port
            [y,k_y] = max(y_prop);
        end;
        Prop_vect = zeros(1, Nbr_prop);
        Prop_vect(k_y) = 1;
    case 3, % 2 pods, only the pod inside the turn rotates
        if turn_sign == -1,
        % turn to starboard
            [y,k_y] = max(y_prop);
        else
        % turn to port
            [y,k_y] = min(y_prop);
        end;
        Prop_vect = zeros(1, Nbr_prop);
        Prop_vect(k_y) = 1;
end;

% Initial point for velocity : Linear motion data
V_lin = Equil_level_ice(P_D, 0, false, false);


% System definition
FCT_turning_P_D_max = @(calc_turning_P_D) Turning_circle(...
                    calc_turning_P_D(1), R_turning, P_D,...
                    calc_turning_P_D(2)*Prop_vect, calc_turning_P_D(3),...
                    turn_sign, turn_floes, fig_local);

options = optimset('MaxFunEvals', 1000, 'MaxIter', 200, 'Display', 'off');


% Calculation for a grid of initial points
V_turning_calc      = zeros(3,3);
beta_prop_deg_calc  = zeros(3,3);
beta_drift_deg_calc = zeros(3,3);
flag_calc           = zeros(3,3);

coeff_V_init   = [1 5 10];
beta_prop_init = [1 25 75];
```

```
for i=1:3,
    for j=1:3,
        % disp([num2str(3*(i-1)+j),'/9']) % Counter
        % Initial values
        calc_turning_0 = [ coeff_V_init(j)*max(V_lin, 0.1),...
                           -turn_sign*beta_prop_init(i),...
                           -turn_sign*5];
        % Resolution
        [rslt_turning_P_D_max,fval,exitflag,output] =...
                    fsolve(FCT_turning_P_D_max, calc_turning_0, options);

        V_turning_calc(i,j)      = rslt_turning_P_D_max(1);%Turning vel.
        beta_prop_deg_calc(i,j) = rslt_turning_P_D_max(2);%Pod(s) angle
        beta_drift_deg_calc(i,j)= rslt_turning_P_D_max(3);%Drift angle
        flag_calc(i,j)           = exitflag;
    end;
end;

%Analysis of the simulation
V_turning_r      = reshape(V_turning_calc, 1,9);
beta_prop_deg_r  = reshape(beta_prop_deg_calc, 1,9);
beta_drift_deg_r = reshape(beta_drift_deg_calc, 1,9);
flag_r           = reshape(flag_calc, 1,9);

l=1;
while l<=length(V_turning_r),
    if (flag_r(l) <= 0)              ||...
       (abs(beta_prop_deg_r(l)) > 90) ||...
       (V_turning_r(l) < 0),
        V_turning_r(l)      = [];
        beta_prop_deg_r(l)  = [];
        beta_drift_deg_r(l) = [];
        flag_r(l)           = [];
    else
    l=l+1;
    end;
end;
[max_V_diff, l_max_V] = max(V_turning_r);

if ~isempty(V_turning_r),
    V_turning      = V_turning_r(l_max_V);      %Turning velocity
    beta_prop_deg  = beta_prop_deg_r(l_max_V); %Pod(s) angle
    beta_drift_deg  = beta_drift_deg_r(l_max_V);%Drift angle
else
    % Unable to find a turning velocity for the requested power level
    V_turning      = NaN; %Turning velocity
    beta_prop_deg  = NaN; %Pod(s) angle
    beta_drift_deg  = NaN; %Drift angle
end;

end
```

### *D.6.3 Turning_circle.m*

```
function [Sum_vect] = Turning_circle(V_turning, R_turning, P_D,...
                      beta_prop_deg, beta_drift_deg, turn_sign, turn_floes
                          ,...
                      fig_local)
%
% Calculation of the resulting forces and moment on a ship in turning
% motion. This function is used to determine the equilibrium point (when
% resulting forces and moment are zero) for a given turning configuration:
% - Calc_circle_V.m    : R_turning and V_turning given,
%                        P_D, drift angle and pod(s) angle to be calculated
% - Calc_circle_P_D.m : R_turning and P_D given,
%                        V_turning, drift angle and pod(s) angle to be
%                        calculated
%
% INPUTS :
% V_turning          [m/s]      Velocity of the ship's centre of gravity
% R_turning          [m]        Turning radius
% P_D                [W]        Ship's delivered power
% beta_prop_deg      [deg]      Pod(s) orientation angle
% beta_drift_deg     [deg]      Ship's drift angle
% turn_sign          [-]        -1 for turn to starboard,+1 for turn to port
% turn_floes         [-]        0 for turn in level ice, 1 for turn in floes
% fig_local          [Bool]     Activate/Deactivate figures plotting for
%                               hull waterline estimation
% OUTPUT:
% Sum_vect           [N,N,N*m] Resulting forces along x,y and moment
%

global Ice_Water_data
global Ship_data
global Prop_data
global Structure_data
global Other_data

V_x =              V_turning*cos(beta_drift_deg*pi/180); %Forward velocity
V_y =             -V_turning*sin(beta_drift_deg*pi/180); %Lateral velocity
r   = -turn_sign*V_turning/R_turning;                    %Yaw velocity

beta_prop_rad = beta_prop_deg*pi/180;

% COORDINATE SYSTEM :
% x parallel to the plane of symmetry, pointing towards bow
% y perpendicular to x, pointing towards starboard
% z perpendicular to x and y, pointing downwards


% READING ICE WATER PARAMETERS
H_ice     = Ice_Water_data{2};% Ice thickness

% READING SHIP PARAMETERS
L_pp       = Ship_data{2}(1);  % Ship's length
B          = Ship_data{3};     % Ship's breadth
mu         = Ship_data{12};    % Friction coefficient
k_turning  = Ship_data{20};    % Coefficient for turning in ice
rudder     = Ship_data{21}(3); % True if ship equipped with rudder, False
                               % otherwise

% READING PROPELLER DATA
Nbr_prop = Prop_data{3};      % Number of propellers
x_prop   = Prop_data{9}(1,:);% x-coord. of propellers location from aft pp
y_prop   = Prop_data{9}(2,:);% y-coord. of propellers location from aft pp

% READING OTHER PARAMETERS
g         = Other_data{2};


% Hypothesis : ship is turning around the centre of gravity
x_CG = L_pp/2; %from aft pp


%%%% ICE RESISTANCE ON THE FORE PART %%%%

% ELEMENTS
%-Discretization of the bow into four elements
y_s = [-3/4,   -2/4, -1/4,    0,   0, 1/4, 2/4,   3/4]*B/2;%start y-coord.
y_e = [-15/16, -3/4, -2/4, -1/4, 1/4, 2/4, 3/4, 15/16]*B/2;%end y-coord.
y_c = (y_s+y_e)/2; % y-coordinate of the center of the element
x_c = zeros(1, 5);
for n=1:length(y_c),
  % x-coordinate of the center of the element, from aft
  x_c_aft = Hull_geom(y_c(n), fig_local);
    % x-coordinate of the center of the element, from CG
    x_c(n) = x_c_aft - x_CG;
end;

%-Velocity
V_x_el = V_x;
V_y_el = V_y*ones(1,length(y_c))+r*x_c;
V_el   = [sqrt(V_x_el^2+turn_sign*V_y_el(1:4).^2)...
          sqrt(V_x_el^2-turn_sign*V_y_el(5:8).^2)];
% If V_el is non real, it means that the element does not break ice
for n=1:length(V_el),
    if ~isreal(V_el(n)),
        V_el(n) = 0;
    end;
end;

%-Local drift angle at the elements
beta_el_rad = [-atan(V_y_el(1:4)/V_x_el) atan(V_y_el(5:8)/V_x_el)];

beta_el_arm_rad = -[atan(V_y_el(1:4)/V_x_el) atan(V_y_el(5:8)/V_x_el)];

%-Lever arm of the ice force on the elements (regards to the CG)
l_If = zeros(1, length(y_c)); % Lever arm

for n=1:length(y_c),
   % l_If positive when producing a positive clockwise moment(R_ice > 0)
  if abs(beta_el_rad(n)) > 1e-6,
        y_la = (x_c(n)+y_c(n)*cot(beta_el_arm_rad(n)))...
               /(cot(beta_el_arm_rad(n))+1/cot(beta_el_arm_rad(n)));
        x_la = -turn_sign*y_la/cot(beta_el_arm_rad(n));
        l_If(n) = sign(y_la)*sqrt(x_la^2+y_la^2);
  else
        l_If(n) = y_c(n);
  end
end;


%STEM
%-Velocity
x_c_stem = L_pp - x_CG;
V_x_stem = V_x;
V_y_stem = V_y + r*x_c_stem;
V_stem   = sqrt(V_x_stem^2+V_y_stem^2);
%-Local drift angle
beta_stem_arm_rad = -atan(V_y_stem/V_x_stem);
%-Lever arm
% l_If_stem positive when producing a positive clockwise moment(R_ice > 0)
if abs(beta_stem_arm_rad) > 1e-6,
  y_la = x_c_stem/(cot(beta_stem_arm_rad)+1/cot(beta_stem_arm_rad));
  x_la = -turn_sign*y_la/cot(beta_stem_arm_rad);
    l_If_stem = sign(y_la)*sqrt(x_la^2+y_la^2);
else
  l_If_stem = 0;
end;


%RESISTANCE DUE TO POTENTIAL ENERGY LOSS
%Velocity, Local drift angle and Lever arm
Cov_B             = Ice_Water_data{10};  % Ship bottom ice coverage
T                 = Ship_data{4};        % Ship's draft
angles_mean       = Ship_data{11};       % phi, alpha, psi mean values
angles_mean_rad   = angles_mean*pi/180;

A_u     = max(0, B*(Cov_B*L_pp-T/tan(angles_mean_rad(1)))...
                 -0.25*B/tan(angles_mean_rad(2))));%Ice covered bottom area
A_f     = B*T*sqrt((1/sin(angles_mean_rad(1)))^2 ...
                 +(1/tan(angles_mean_rad(2)))^2);  %Bow area
x_c_u = (1-Cov_B)*L_pp + ((L_pp-T/tan(angles_mean_rad(1)))...
          - (1-Cov_B)*L_pp)/2 - x_CG;
x_c_f = L_pp - x_CG - T/(2*tan(angles_mean_rad(1)));

%x_c_p =  (A_u/(A_u+A_f))*x_c_u + (A_f/(A_u+A_f))*x_c_f;%Theoretical value
x_c_p =  ((0.5*A_u+0*A_f)/(A_u+A_f))*x_c_u...
         + ((0.5*A_u+1*A_f)/(A_u+A_f))*x_c_f;%Empirically determined value

%-Velocity
V_x_p = V_x;
V_y_p = V_y + r*x_c_p;
V_p   = sqrt(V_x_p^2+V_y_p^2);
%-Local drift angle
beta_p_arm_rad = -atan(V_y_p/V_x_p);
%-Lever arm
% l_If_p positive when producing a positive clockwise moment (R_ice > 0)
if abs(beta_p_arm_rad) > 1e-6,
  y_la = x_c_p/(cot(beta_p_arm_rad)+1/cot(beta_p_arm_rad));
  x_la = -turn_sign*y_la/cot(beta_p_arm_rad);
    l_If_p = sign(y_la)*sqrt(x_la^2+y_la^2);
else
  l_If_p = 0;
end;


%FRICTIONAL RESISTANCE
%Velocity, Local drift angle and Lever arm
% => idem potential energy loss under the bottom
x_c_fr = x_c_u;
%-Velocity
V_x_fr = V_x;
V_y_fr = V_y + r*x_c_fr;
V_fr   = sqrt(V_x_fr^2+V_y_fr^2);
%-Local drift angle
beta_fr_arm_rad = -atan(V_y_fr/V_x_fr);
%-Lever arm
% l_If_fr positive when producing a positive clockwise moment (R_ice > 0)
if abs(beta_fr_arm_rad) > 1e-6,
  y_la = x_c_fr/(cot(beta_fr_arm_rad)+1/cot(beta_fr_arm_rad));
  x_la = -turn_sign*y_la/cot(beta_fr_arm_rad);
    l_If_fr = sign(y_la)*sqrt(x_la^2+y_la^2);
else
  l_If_fr = 0;
end;


% Ice resistance of the ship
[R_c, R_b, R_p, R_f, r_w, R_ow] = Discr_Ice_resistance(V_stem,...
```

```
                                                        beta_el_rad);
if turn_floes, %turn in floes
    d_floe_target = Structure_data{6};% Target floe size
    k_channel    = Ship_data{18};    % Channel width to ship's beam ratio
    W_channel     = B*k_channel;      % Icebreaker channel width
    d_floe_pass1 = 2*d_floe_target;
    C_ice_pass1  = (d_floe_pass1^2)...
                   /(d_floe_pass1^2 + d_floe_pass1*W_channel);

    R_floes = Ice_resistance_floes(V_stem, d_floe_pass1, C_ice_pass1);
    R_lvl   = Ice_resistance(V_stem, false);

    % Reduction of the resistance by the ratio R_floes/R_lvl
    R_c  = (R_floes/R_lvl)*R_c;
    R_b  = (R_floes/R_lvl)*R_b;
    R_p  = (R_floes/R_lvl)*R_p;
    R_f  = (R_floes/R_lvl)*R_f;
    r_w  = (R_floes/R_lvl)*r_w;
    R_ow = (R_floes/R_lvl)*R_ow;
end;

% Velocity influence
R_c_If = R_c*(1+1.4*V_stem/sqrt(g*H_ice));%Crushing at the stem
R_b_If = R_b'.*(1+1.4*V_el/sqrt(g*H_ice));%Ice bending along the hull
R_p_If = R_p*(1+9.4*V_p/sqrt(g*L_pp));    %Res. due to potent. energy loss
R_f_If = R_f*(1+9.4*V_fr/sqrt(g*L_pp));   %Frictional resistance
R_w_If = r_w*V_stem^2;                    %Wedge resistance
R_ow_If= (2/3)*R_ow;                      %Open water resistance

%Check for minimal ice resistance compared to open-water resistance
R_tot_If = R_c_If + sum(R_b_If) + R_p_If + R_f_If + R_w_If + R_ow_If;
if R_tot_If < R_ow,
  R_c_If  = 0;
    R_b_If  = zeros(1,length(R_b_If));
  R_p_If  = 0;
  R_f_If  = 0;
    R_w_If  = 0;
    R_ow_If = R_ow;
end;

for n=1:length(V_el),
    %When no velocity relatively to the ice, no breaking resistance
    if V_el(n) == 0,
        R_b_If(n) = 0;
    end;
end;

% Forces along x and y : positive in positive x and y directions
% Moment : positive clockwise
F_If_tot_x = dot(-R_b_If, cos(beta_el_arm_rad))...
             - R_c_If*cos(beta_stem_arm_rad)...
             - R_p_If*cos(beta_p_arm_rad)...
             - R_f_If*cos(beta_fr_arm_rad)...
             - (R_w_If+R_ow_If)*cos(beta_stem_arm_rad);

F_If_tot_y = dot(R_b_If, sin(beta_el_arm_rad))...
             + R_c_If*sin(beta_stem_arm_rad)...
             + R_p_If*sin(beta_p_arm_rad)...
             + R_f_If*sin(beta_fr_arm_rad)...
             + (R_w_If+R_ow_If)*sin(beta_stem_arm_rad);

M_F_If_tot = dot(R_b_If, l_If)...
             + R_c_If*l_If_stem...
             + R_p_If*l_If_p...
             + R_f_If*l_If_fr...
             + (R_w_If+R_ow_If)*l_If_stem;


%%%%% ICE RESISTANCE ON THE AFT PART %%%%%
% Normal force on the aft side
N_lat_add = -k_turning*turn_sign*R_tot_If*(L_pp/R_turning);
% Lateral force due to friction on the aft side
T_lat_add = -mu*abs(N_lat_add);
% Length of the side portion where normal force is applied
a = L_pp/4;
% Lever arms due to normal and lateral forces on the aft part
l_N_lat_add = -(x_CG - a/2);
l_T_lat_add = -turn_sign*B/2;


% Forces along x and y : positive in positive x and y directions
% Moment : positive clockwise
F_Ir_tot_x =  T_lat_add;
F_Ir_tot_y =  N_lat_add;
M_F_Ir_tot =  N_lat_add*l_N_lat_add + T_lat_add*l_T_lat_add;



%%%%% PROPELLER(S) THRUST %%%%%
V_prop = zeros(1, Nbr_prop);
T_net  = zeros(1, Nbr_prop);
N_calc = zeros(1, Nbr_prop);
if rudder, % If ship equipped with fixed propeller and rudder
    F_rud_X          = zeros(1, Nbr_prop);
    F_rud_Y          = zeros(1, Nbr_prop);
    x_F_rud_aft_pp   = zeros(1, Nbr_prop);
else % Ship equipped with pod(s)
    l_prop = zeros(1, Nbr_prop);
end;
x_prop_cg = x_prop - x_CG; %x coordinate of propellers location from CG
for k_prop = 1:Nbr_prop,
    V_prop_x = V_x - r*y_prop(k_prop);
    V_prop_y = V_y + r*x_prop_cg(k_prop);
    V_prop(k_prop)   = sqrt(V_prop_x^2+V_prop_y^2);

    if P_D>0,
        [T_net(k_prop),N_calc(k_prop)] = Thrust_calc_1prop(...
            P_D/Nbr_prop, V_prop(k_prop), fig_local);
    else
        T_net(k_prop) = 0;
        N_calc(k_prop) = 0;
    end;

    if rudder, % If ship equipped with fixed propeller and rudder
        % If ship has fixed propeller(s) and rudder(s), calculation of the
        % corresponding net thrust amplitude, direction and application
        % point.
        F_rud_X_0 = Rudder_force_map(V_turning, N_calc(k_prop), 0);
        T_net_0 =Thrust_calc_1prop(P_D/Nbr_prop,V_prop(k_prop),fig_local);

        [F_rud_X(k_prop), F_rud_Y(k_prop), x_F_rud_aft_pp(k_prop)] = ...
                        Rudder_force_map(V_turning, N_calc(k_prop),...
                        beta_prop_rad(k_prop));
        % Initial rudder drag correction
        F_rud_X(k_prop) = F_rud_X(k_prop) - (F_rud_X_0-T_net_0);
    end;
    % Lever arm of the thrust force (regards to the CG)
    % l_prop positive when producing a positive counterclockwise moment
    % (T_net > 0)
    if abs(beta_prop_rad(k_prop)) > 1e-6,
        y_la = (x_prop_cg(k_prop)+y_prop(k_prop)...
             *cot(beta_prop_rad(k_prop)))...
             /(cot(beta_prop_rad(k_prop))+1/cot(beta_prop_rad(k_prop)));
        x_la = -turn_sign*y_la/cot(beta_prop_rad(k_prop));
        l_prop(k_prop) = sign(y_la)*sqrt(x_la^2+y_la^2);
    else
        l_prop(k_prop) = y_prop(k_prop);
    end
end;
% Forces along x and y : positive in positive x and y directions
% Moment : positive counterclockwise
if ~rudder,
    T_net_tot_x = dot(T_net, cos(beta_prop_rad));
    T_net_tot_y = dot(T_net, -sin(beta_prop_rad));
    M_T_net_tot = dot(T_net, l_prop);
else
    T_net_tot_x = sum(F_rud_X);
    T_net_tot_y = sum(F_rud_Y);
    M_T_net_tot = dot(F_rud_Y, x_CG - x_F_rud_aft_pp);
end;

%%%%% FORCES AND MOMENT EQUILIBRIUM %%%%%
% Longitudinal force equilibrium equation
Sum_Fx = T_net_tot_x + F_If_tot_x + F_Ir_tot_x;%(+ in positive x direct.)
% Longitudinal force equilibrium equation
Sum_Fy = T_net_tot_y + F_If_tot_y + F_Ir_tot_y;%(+ in positive y direct.)
% Moment equilibrium equation
Sum_M  = M_T_net_tot - M_F_If_tot - M_F_Ir_tot;%(+ counterclockwise)

% Output
Sum_vect = [Sum_Fx Sum_Fy Sum_M];

end
```

### *D.6.4 Hull_geom.m*

```
function x_wl = Hull_geom(y, fig)

% Calculation of bow geometry on the waterline at a given y position. This
% function uses an interpolation of the bow contour data.
%
% INPUTS :
% y                 [m]     y position, from centerline
% fig               [Bool]  Activate/Deactivate figure plotting
%
% OUTPUT:
% x_wl              [m]     x coordinate at the waterline, from aft pp
%

global Ship_data

% READING SHIP PARAMETERS
B_ship   = Ship_data{3};  % Ship's breadth
X_WL_bow = Ship_data{19}; % Discrete values of X at the bow on the WL

% Corresponding Y values at the bow on the waterline
Y_WL_bow = [0, 1/4, 2/4, 3/4, 15/16]*B_ship/2;

% Outputs calculation
x_wl     = interp1(Y_WL_bow, X_WL_bow, abs(y));


% REPRESENTATION OF APPROXIMATED WATERLINE
if fig,
   figure('Position', [0 0 600 200]),
   Ref      = Ship_data{1};    % Reference
   dy_repr = max(Y_WL_bow)/20;

   % Outputs calculation
   y_repr = 0:dy_repr:max(Y_WL_bow);
   x_wl_repr   = interp1(Y_WL_bow,X_WL_bow, y_repr);

   plot(X_WL_bow, Y_WL_bow,    's','Color','r','MarkerFaceColor','r');
   hold on,
   plot(x_wl_repr, y_repr,'Color', 'b');


   legend('Data points', 'Approximations');
   xlabel('x [m]', 'Fontsize', 12)
   ylabel('y [m]', 'Fontsize', 12)
   title(['Waterline',' approximation - Reference : ', Ref], 'Fontsize'
         ,12)
   axis([min(X_WL_bow)-10 max(X_WL_bow)+10 0 1.2*max(Y_WL_bow)])
end;

end
```

### *D.6.5 Rudder_force_map.m*

```
function [F_rud_X, F_rud_Y, x_F_rud_aft_pp] = Rudder_force_map(...
                                        V_turning, N_calc, beta_prop_rad)
%
%
% In the case of a ship equipped with fixed propeller and rudder, this
% function computes the resulting force amplitude, direction and
% application point for any given ship velocity, revolution rate and
% rudder angle.
%
% This function considers the creation of a map of rudder forces, for
% faster calculation in an iterative process with constant velocity.
%
% Rem : This function requires the input files "prop.txt", "rpsim.txt" and
%       "rudder.txt" to be filled with the appropriated data.
%
% INPUTS :
% V               [m/s]     Ship's velocity
% N_calc          [1/s]     Propeller revolution rate
% delta_rad       [rad]     Rudder angle
%
% OUTPUTS:
% F_rud_X         [N]       Total net thrust in X direction
% F_rud_Y         [N]       Total net thrust in Y direction
% x_F_rud_aft_pp  [m]       Total net thrust application point, from aft PP
%

global Ship_data



% Map of rudder forces
map_file = ['Input_data\Rudder\Rudder_force_map_V',...
            num2str(V_turning, 4),'.mat'];
if exist(map_file, 'file'),
    %Try to load a pre-existing map
    %disp('Loading the pre-existing map of rudder forces')
    load(map_file)
else
    % Creation of the map of the rudder forces
    disp('Creation of a map of rudder forces')
    P_D_max_MW = Ship_data{14}(1); % Ship's max delivered power (! in MW)


    V   = V_turning; %Neglecting the effect of the drift angle

    P_D_vect=(.05*P_D_max_MW*1e6):(.05*P_D_max_MW*1e6):(4*P_D_max_MW*1e6);
    delta_deg_vect  = 0:5:90;

    N_calc_vect         = zeros(1, length(P_D_vect));
    F_rud_X_mat         = zeros(length(P_D_vect), length(delta_deg_vect));
    F_rud_Y_mat         = zeros(length(P_D_vect), length(delta_deg_vect));
    x_F_rud_aft_pp_mat = zeros(length(P_D_vect), length(delta_deg_vect));


    for i = 1:length(P_D_vect),
        disp([num2str(i), '/', num2str(length(P_D_vect))])

        [T_net, N_calc_vect(i)] = Thrust_calc_1prop(P_D_vect(i),V,false);
        delta_rad_vect = delta_deg_vect*pi/180;

        for j = 1:length(delta_rad_vect),
           [F_rud_X_mat(i,j),F_rud_Y_mat(i,j),x_F_rud_aft_pp_mat(i,j)]=...
                    Rudder_force(V, N_calc_vect(i), delta_rad_vect(j));
        end;
    end;
    delta_deg_vect =horzcat(-delta_deg_vect((end:(-1):2)),delta_deg_vect);
    F_rud_X_mat         =horzcat(F_rud_X_mat(:,(end:(-1):2)),F_rud_X_mat);
    F_rud_Y_mat         =horzcat(-F_rud_Y_mat(:,(end:(-1):2)),F_rud_Y_mat);
    x_F_rud_aft_pp_mat =horzcat(x_F_rud_aft_pp_mat(:,(end:(-1):2)),...
                                                    x_F_rud_aft_pp_mat);
    % Saving the matrix to a file
     save(map_file, 'N_calc_vect', 'delta_deg_vect', 'F_rud_X_mat',...
                    'F_rud_Y_mat', 'x_F_rud_aft_pp_mat')
end;
% Calculation of the results from the map
if (N_calc > min(N_calc_vect)) && (N_calc < max(N_calc_vect)) &&...
   ((beta_prop_rad*180/pi) > min(delta_deg_vect)) &&...
   ((beta_prop_rad*180/pi) < max(delta_deg_vect)),
    F_rud_X = interp2(N_calc_vect, delta_deg_vect', F_rud_X_mat',...
                      N_calc, beta_prop_rad*180/pi);
    F_rud_Y = interp2(N_calc_vect, delta_deg_vect', F_rud_Y_mat',...
                      N_calc, beta_prop_rad*180/pi);
    x_F_rud_aft_pp = interp2(N_calc_vect, delta_deg_vect',...
                     x_F_rud_aft_pp_mat', N_calc, beta_prop_rad*180/pi);
else
    % Calculation for the point outside the map
    if abs(beta_prop_rad) > pi/2,
        %disp('case 1')
        F_rud_X = NaN;
        F_rud_Y = NaN;
        x_F_rud_aft_pp = NaN;
    else
        %disp('case 2')
        beta_prop_rad1 = floor(beta_prop_rad*100)/100-0.0001;
        beta_prop_rad2 = ceil(beta_prop_rad*100)/100+0.0001;

        [F_rud_X1, F_rud_Y1, x_F_rud_aft_pp1] = Rudder_force(V_turning,...
                                                N_calc, beta_prop_rad1);
        [F_rud_X2, F_rud_Y2, x_F_rud_aft_pp2] = Rudder_force(V_turning,...
                                                N_calc, beta_prop_rad2);

        F_rud_X = interp1([beta_prop_rad1 beta_prop_rad2],...
                          [F_rud_X1 F_rud_X2],...
                           beta_prop_rad);
        F_rud_Y = interp1([beta_prop_rad1 beta_prop_rad2],...
                          [F_rud_Y1 F_rud_Y2],...
                           beta_prop_rad);
        x_F_rud_aft_pp =interp1([beta_prop_rad1 beta_prop_rad2],...
                                [x_F_rud_aft_pp1 x_F_rud_aft_pp2],...
                                 beta_prop_rad);
    end;
end;
```

## D.7 Ice Management Strategy and Simulation

### *D.7.1 Ice_management_linear.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                          SIMULATION OF THE                                    %
%                     LINEAR ICE MANAGEMENT TECHNIQUE                           %
%                                                                               %
%                            Quentin HISETTE                                    %
%                                                                               %
%                              2013-2014                                        %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc

fig_local       = false;%[Bool] Activate/Deactivate local figures plotting
fig_main        = true; %[Bool] Activate/Deactivate main figures plotting
L_area_analysis = false;%[Bool] Activate/Deactivate param. study of L_area
movie_rec       = false;%[Bool] Activate/Deactivate movie recording

% READING INPUT FILES
disp('Reading input files')
Input_reading;
global Ship_data
global Structure_data
global Ice_drift_data
global Other_data


% READING STRUCTURE PARAMETERS
Ref           = Structure_data{1};    % Reference
L_struct    = Structure_data{2}(1); % Structure length
B_struct      = Structure_data{2}(2); % Structure breadth
D_struct      = Structure_data{2}(3); % Structure diameter
Psi_struct    = Structure_data{3};    % Structure course
t_cr      = Structure_data{4};    % Critical distance to level ice
R_excl        = Structure_data{5};    % Ships exclusion zone radius
d_floe_target = Structure_data{6};    % Target floe size

% READING ICE DRIFT DATA
V_drift       = Ice_drift_data{2};  % Drift velocity
alpha_drift   = Ice_drift_data{3};  % Drift direction

% READING SHIP PARAMETERS
B_ship        = Ship_data{3};     % Ship's breadth
P_D_max_MW    = Ship_data{14}(1); % Ship's max delivered power (! in MW)
k_channel   = Ship_data{18};    % Channel width to ship's beam ratio
prop_mode     = Ship_data{21}(1); % Propeller(s) rotating mode :
                        % 1: All propellers are rotating
                        % 2: 2 pods, only the pod outside the turn rotates
                        % 3: 2 pods, only the pod inside the turn rotates

advance_mode  = Ship_data{21}(1); % Type of advance along the trajectory :
                        % 1: Keeping same power as in linear motion on the
                        %    whole trajectory, V decreases in turns and
                        %    increases in floes.
                        % 2: Keeping same velocity as in linear motion on
                        %    the whole trajectory, P_D increases in turns
                        %    (< P_D_max) and decreases in floes.

% READING STRUCTURE PARAMETERS
double_pass = Structure_data{7};% True if double pass management technique

% READING OTHER PARAMETERS
dt            = Other_data{3};  % Time step

if V_drift == 0,
    disp('ERROR : Zero drift velocity.')
    error('SIMULATION ABORTED');
end;


%% ICE MANAGEMENT CALCULATIONS
disp('Calculation of managed area width')

Psi_struct_rad  = mod(Psi_struct,360)*pi/180;
alpha_drift_rad = mod(alpha_drift,360)*pi/180;

% Icebreaker channel width
W_channel = B_ship*k_channel;

% Width of the trajectory
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    W_traj = D_struct + 2*t_cr - W_channel;
elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    W_traj = abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
        + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
        + 2*t_cr...
        - W_channel;
end;

% Width of the managed area
W_area_m = W_traj + W_channel;


% Number of passes of the ice management trajectory (linear method)
N_pass = ceil(W_traj/(W_channel+d_floe_target))+1;

if double_pass && N_pass < 5,
    % Minimum N_pass value is 5 for double pass ice management
    floe_size_max_double_pass = W_area_m/(4-1)-W_channel;
    disp(['ERROR : Unable to use a double pass ice management. Area ',...
          'to be managed not wide enough. Possible solutions :'])
    disp('- Use single pass ice management,')
    disp(['- Reduce target floe size below ',...
          num2str(floe_size_max_double_pass,3),' m.'])
    error('SIMULATION ABORTED')
end;


% Parameters for resistance in floes
if double_pass,
    d_floe_pass1 = 2*d_floe_target;
    C_ice_pass1  = (d_floe_pass1^2)...
        /(d_floe_pass1^2 + d_floe_pass1*W_channel);
end;

%% PARAMETRIC STUDY OF L_area

if L_area_analysis,

    P_D_vect      = ((P_D_max_MW/5):(P_D_max_MW/10):(2*P_D_max_MW))*1e6;
    V_drift_vect = [0.1  0.25:0.05:1];

    L_area_mat = zeros(length(P_D_vect), length(V_drift_vect));


    V_drift_calc    = [0, V_drift, -V_drift];
    V_lvl_ice_vect  = zeros(length(P_D_vect),3);
    V_fl_vect       = zeros(length(P_D_vect),3);
    V_mean_ice_vect = zeros(length(P_D_vect),3);

    V_lvl_nodrift_vect   = zeros(length(P_D_vect),1);
    V_lvl_updrift_vect   = zeros(length(P_D_vect),1);
    V_lvl_downdrift_vect = zeros(length(P_D_vect),1);
    V_lvl_turning_vect   = zeros(length(P_D_vect),1);

    V_fl_turning_vect    = zeros(length(P_D_vect),1);

    V_mean_nodrift_vect    = zeros(length(P_D_vect),1);
    V_mean_updrift_vect    = zeros(length(P_D_vect),1);
    V_mean_downdrift_vect  = zeros(length(P_D_vect),1);
    V_mean_turning_vect    = zeros(length(P_D_vect),1);

    for j = 1:length(P_D_vect),
        disp([num2str(j), '/', num2str(length(P_D_vect))])
        disp_eq = false;
        for k=1:3,
            V_lvl_ice_vect(j,k) = Equil_level_ice(P_D_vect(j),...
                V_drift_calc(k), disp_eq, fig_local);
        end;

        V_lvl_kts_vect = V_lvl_ice_vect*1.94384449;

        V_lvl_nodrift_vect(j)   = V_lvl_ice_vect(j,1);
        V_lvl_updrift_vect(j)   = V_lvl_ice_vect(j,2);
        V_lvl_downdrift_vect(j) = V_lvl_ice_vect(j,3);

        % Required turning radius
        R_turning = (d_floe_target+W_channel)/2;
        if double_pass,
            R_turning = 2*R_turning;
        end;
        turn_sign = +1;  % +1 for turn to port (arbitrary choice)
                         %

        if advance_mode == 1, %Turning at same power as linear motion
            P_D_turning = P_D_vect(j);
            V_lvl_turning_vect(j) = Calc_circle_P_D(R_turning,...
                    P_D_turning, turn_sign, prop_mode, false, fig_local);

        elseif advance_mode == 2,%Turning at same velocity as lin. motion
            V_lvl_turning_vect(j) = V_lvl_nodrift_vect(j);
            P_D_turning = Calc_circle_V(R_turning, V_lvl_turning_vect,...
                    turn_sign, prop_mode, false, fig_local);
        else
            disp('Unknown turning mode.')
            error('SIMULATION ABORTED')
        end;

        if double_pass,
            if advance_mode == 1,% Advance in floes at same power as
                                 % linear motion
                for k=1:3, %Linear motion in floes
                    V_fl_vect(j,k) = Equil_floes(P_D_vect(j),...
                       V_drift_calc(k), d_floe_pass1, C_ice_pass1,...
                       disp_eq, fig_local);
                end;
                % Turning in floes
                P_D_turning_fl = P_D_vect(j);
                V_fl_turning_vect(j) = Calc_circle_P_D(R_turning,...
                   P_D_turning, turn_sign, prop_mode, true, fig_local);

            elseif advance_mode == 2,% Advance in floes at same velocity
                                     % as linear motion
                V_fl_vect(j,:) = V_lvl_ice_vect(j,:);

                V_fl_turning_vect(j) = V_lvl_nodrift_vect(j);
                P_D_turning_fl = Calc_circle_V(R_turning,...
                        V_fl_turning_vect, turn_sign, prop_mode, true,...
                        fig_local);
            end

            V_mean_ice_vect(j,:)      = ...
                        2./(1./V_lvl_ice_vect(j,:) + 1./V_fl_vect(j,:));
            V_mean_nodrift_vect(j)    = V_mean_ice_vect(j,1);
            V_mean_updrift_vect(j)    = V_mean_ice_vect(j,2);
            V_mean_downdrift_vect(j) = V_mean_ice_vect(j,3);

            V_mean_turning_vect(j)    = ...
                      2/(1/V_lvl_turning_vect(j) + 1/V_fl_turning_vect(j));
        end;

        for l = 1:length(V_drift_vect),
            if ~double_pass,
                %single pass
                L_area_mat(j,l) = V_drift_vect(l)*(...
                  (-N_pass*R_turning)/V_lvl_updrift_vect(j)...
                 +(-N_pass*R_turning)/V_lvl_downdrift_vect(j)...
                 +(W_traj-2*R_turning)/V_lvl_nodrift_vect(j)...
                 +N_pass*pi*R_turning/V_lvl_turning_vect(j))...
                 /(1-V_drift_vect(l)*(N_pass/(2*V_lvl_updrift_vect(j))+...
                   N_pass/(2*V_lvl_downdrift_vect(j))));
            else
                %double pass
                L_area_mat(j,l) = V_drift_vect(l)*(...
                (-N_pass*R_turning)/V_mean_updrift_vect(j)...
                +(-N_pass*R_turning)/V_mean_downdrift_vect(j)...
                +2*(W_traj-3*R_turning)/V_mean_nodrift_vect(j)...
                +N_pass*pi*R_turning/V_mean_turning_vect(j))...
                /(1-V_drift_vect(l)*(N_pass/(2*V_mean_updrift_vect(j))+...
                N_pass/(2*V_mean_downdrift_vect(j))));
            end;
            if L_area_mat(j,l) < 0,
                L_area_mat(j,l) = NaN;
            end;
        end;
    end;

    % FIGURES
    % L_area as a function of P_D
    figure('position', [0 0 900 360]),
    subplot(1,2,1)
    plot(P_D_vect/1e6, L_area_mat(:,1),  'b')
```

```
    hold on,
    plot(P_D_vect/1e6, L_area_mat(:,2), 'r')
    hold on,
    plot(P_D_vect/1e6, L_area_mat(:,7), 'g')
    hold on,
    plot(P_D_vect/1e6, L_area_mat(:,12), 'k')
    hold on,
    plot(P_D_vect/1e6, L_area_mat(:,17), 'm')
    h_leg=legend(['$V_{drift} = $',num2str(V_drift_vect(1)),'$\,m/s$'],...
        ['$V_{drift} = $', num2str(V_drift_vect(2)), '$\,m/s$'],...
        ['$V_{drift} = $', num2str(V_drift_vect(7)), '$\,m/s$'],...
        ['$V_{drift} = $', num2str(V_drift_vect(12)), '$\,m/s$'],...
        ['$V_{drift} = $', num2str(V_drift_vect(17)), '$\,m/s$'],...
        'Orientation', 'Vertical');
    set(h_leg, 'FontSize', 9, 'Interpreter', 'latex')
    xlabel('$P_D$ $[MW]$', 'Fontsize', 12, 'Interpreter', 'latex')
    ylabel('$L_{area}$ $[m]$', 'Fontsize', 12, 'Interpreter', 'latex')
    title('Influence of delivered power',...
          'Fontsize', 12)
    axis([0 35 0 1500])

    % L_area as a function of V_drift
    subplot(1,2,2)
    plot(V_drift_vect, L_area_mat(4,:), 'b')
    hold on,
    plot(V_drift_vect, L_area_mat(7,:), 'r')
    hold on,
    plot(V_drift_vect, L_area_mat(9,:), 'g')
    hold on,
    plot(V_drift_vect, L_area_mat(11,:), 'k')
    hold on,
    plot(V_drift_vect, L_area_mat(13,:), 'm')
    h_leg = legend(['$P_{D} = $', num2str(P_D_vect(4)/1e6), '$\,MW$'],...
        ['$P_{D} = $', num2str(P_D_vect(7)/1e6), '$\,MW$'],...
        ['$P_{D} = $', num2str(P_D_vect(9)/1e6), '$\,MW$'],...
        ['$P_{D} = $', num2str(P_D_vect(11)/1e6), '$\,MW$'],...
        ['$P_{D} = $', num2str(P_D_vect(13)/1e6), '$\,MW$'],...
        'Orientation', 'Vertical',...
        'Location', 'NorthWest');
    set(h_leg, 'FontSize', 9, 'Interpreter', 'latex')
    xlabel('$V_{drift}$ $[m/s]$', 'Fontsize', 12, 'Interpreter', 'latex')
    ylabel('$L_{area}$ $[m]$', 'Fontsize', 12, 'Interpreter', 'latex')
    title('Influence of drift speed',...
          'Fontsize', 12)
    axis([0 0.75 0 1500])
end;



%% FINDING A GOOD VALUE FOR L_area

ex_l_area = 1.25; % L_area_90 (with 90% of P_D) has to be less than
                  % ex_l_area*L_area

disp('Calculation of the length of managed area:')
[L_area, P_D, V_lin_ice, R_turning, V_turning, beta_prop_deg,...
                beta_drift_deg, P_D_turning] = L_area_fct(V_drift,...
                ex_l_area, W_channel, d_floe_target, W_traj, prop_mode,...
                advance_mode, fig_local);

% IF INSUFFICIENT POWER, COMPUTE MIN MANAGEABLE FLOE SIZE AND MAXIMUM
% MANAGEABLE AREA
% - MINIMUM MANAGEABLE FLOE SIZE
if P_D > P_D_max_MW*1e6,
    % The number of passes must be decreased
    N_pass = ceil(W_traj/(W_channel+d_floe_target))+1;
    if N_pass >= 3,
        disp('Insufficient ship power.')
        disp('* Calculation of minimum manageable floe size...')
    end;
  P_D_lim_floe = P_D;
    V_lin_ice_lim_floe = V_lin_ice;
  while (P_D_lim_floe > P_D_max_MW*1e6) && (N_pass >= 3),
        N_pass = N_pass-1;
        if double_pass && N_pass < 5,
            % Minimum N_pass value is 5 for double pass ice management
            floe_size_max_double_pass = W_area_m/(4-1)-W_channel;
            disp(['WARNING : Unable to use a double pass ice ',...
                  'management. Calculation for single pass ice ',...
                  'management.'])
            Structure_data{7} = false;
        end;

        d_floe_target_min = W_traj/(N_pass-1) - W_channel;
    [L_area_lim_floe, P_D_lim_floe, V_lin_ice_lim_floe,...
         R_turning_lim_floe, V_turning_lim_floe, beta_prop_deg_lim_floe,...
         beta_drift_deg_lim_floe, P_D_turning_lim_floe] = L_area_fct(...
                    V_drift, ex_l_area, W_channel, d_floe_target_min,...
                    W_traj, prop_mode, advance_mode, fig_local);
  end;
    V_lin_ice = V_lin_ice_lim_floe;
end;
Structure_data{7} = double_pass;


% - MAXIMUM MANAGEABLE AREA WIDTH
if P_D > P_D_max_MW*1e6,
    % The number of passes must be decreased
    N_pass = ceil(W_traj/(W_channel+d_floe_target))+1;
    if N_pass >= 3,
        disp('* Calculation of maximum manageable area width...')
    end;
  P_D_max_area = P_D;
    V_lin_ice_max_area = V_lin_ice;
  while (P_D_max_area > P_D_max_MW*1e6) && (N_pass >= 3),
        N_pass = N_pass-1;
        if double_pass && N_pass < 5,
            % Minimum N_pass value is 5 for double pass ice management
            floe_size_max_double_pass = W_area_m/(4-1)-W_channel;
            disp(['WARNING : Unable to use a double pass ice ',...
                  'management. Calculation for single pass ice ',...
                  'management.'])
            Structure_data{7} = false;
        end;

        W_area_max = (N_pass-1)*(W_channel+d_floe_target)+W_ch;
        W_traj_max = W_area_max - W_ch;
    [L_area_max_area, P_D_max_area, V_lin_ice_max_area,...
         R_turning_max_area, V_turning_max_area, beta_prop_deg_max_area,...
         beta_drift_deg_max_area, P_D_turning_max_area] = L_area_fct(...
                    V_drift, ex_l_area, W_channel, d_floe_target,...
                    W_traj_max, prop_mode, advance_mode, fig_local);
  end;
    V_lin_ice = V_lin_ice_max_area;
end;


%% COMMENTS ABOUT THE CALCULATION
if P_D <= P_D_max_MW*1e6,
  V_lin_kts = V_lin_ice*1.94384449;
    V_turning_kts = V_turning*1.94384449;
    if P_D >= 0.9*P_D_max_MW*1e6,
        safety_new = 100*(P_D_max_MW*1e6 - P_D)/(P_D_max_MW*1e6);
        disp(['WARNING : Ice management is only possible with ',...
              num2str(safety_new,3),...
              '% of security allowance on the delivered power.'])
    end;

  disp(['The optimal managed area length is ',num2str(L_area, 5), ' m'])
    if double_pass && (advance_mode == 2),
        disp(' - Corresponding delivered power (floes - level ice) : ')
        disp([' * Linear motion: ',num2str(P_D(2)/1e6, 3), ' MW - ',...
              num2str(P_D(1)/1e6, 3), ' MW'])
    else
        disp(' - Corresponding delivered power : ')
        disp([' * Linear motion: ',num2str(P_D(1)/1e6, 3), ' MW'])
    end;
    switch advance_mode,
        case 1, %Keeping same power as in linear motion, V decreases in
                %level ice turns and increases in floes
            disp([' * Turning:          ',...
                  num2str(P_D_turning(1)/1e6,3),...
                  ' MW (same as linear motion)'])
        case 2, %Keeping same velocity as in linear motion, P_D increases
                %level ice turns and decreases in floes
            if P_D_turning == P_D_max_MW*1e6,
                disp([' * Turning:          ',...
                      num2str(P_D_turning(1)/1e6,3),...
                      ' MW (max delivered power)'])
            else
                if ~double_pass
                    disp([' * Turning:          ',...
                          num2str(P_D_turning/1e6,3),...
                          ' MW'])
                else
                    disp([' * Turning:          ',...
                          num2str(P_D_turning(2)/1e6,3),' MW - ',...
                          num2str(P_D_turning(1)/1e6, 3), ' MW'])
                end;
            end;
    end;
    if ~double_pass,
        disp(' - Level ice velocities :')
    else
        disp(' - Mean linear motion velocities :')
    end;
  disp([' * No relative drift : ', num2str(V_lin_ice(1), 3),...
        ' m/s (', num2str(V_lin_kts(1),3),' kts)'])
  disp([' * Updrift :           ', num2str(V_lin_ice(2), 3),...
        ' m/s (', num2str(V_lin_kts(2),3),' kts)'])
  disp([' * Downdrift :         ', num2str(V_lin_ice(3), 3),...
        ' m/s (', num2str(V_lin_kts(3),3),' kts)'])
    switch advance_mode,
        case 1, %Keeping same power as in linear motion, V decreases
            disp([' * Turning :           ', num2str(V_turning, 3),...
                  ' m/s (', num2str(V_turning_kts,3),' kts)'])
        case 2, %Keeping same velocity as in linear motion, P_D increases
            if P_D_turning == P_D_max_MW*1e6,
                disp([' * Turning :           ', num2str(V_turning, 3),...
                      ' m/s (', num2str(V_turning_kts,3),...
                      ' kts) (at max delivered power)'])
            else
               disp([' * Turning :           ', num2str(V_turning, 3),...
                     ' m/s (', num2str(V_turning_kts,3),...
                     ' kts) (same as linear motion)'])
            end
    end;
    disp(['- Turning radius : ', num2str(R_turning,3), ' m'])

    if ~double_pass
        switch prop_mode,
            %Single pass
            case 1, % All propellers are rotating
                disp(['- Propeller(s) angle : +/- ',...
                    num2str(abs(beta_prop_deg),3), ' deg'])
            case 2, % 2 pods, only the pod outside the turn rotates
                disp(['- Propellers angles : ',...
                    num2str(beta_prop_deg,3),...
                    ' deg (outside prop.), ', num2str(0.0,3),...
                    ' deg (inside prop.)'])
            case 3, % 2 pods, only the pod inside the turn rotates
                disp(['- Propellers angles : ', num2str(0.0,3),...
                    ' deg (outside prop.), ', num2str(beta_prop_deg,3),...
                    ' deg (inside prop.)'])
        end;
    else
        %Double pass
        switch prop_mode,
            %Single pass
            case 1, % All propellers are rotating
                disp(['- Propeller(s) angle : +/- ',...
                    num2str(abs(beta_prop_deg(2)),3), ' deg - ',...
                    num2str(abs(beta_prop_deg(1)),3), ' deg ',...
                    '(floes - level ice)'])
            case 2, % 2 pods, only the pod outside the turn rotates
                disp(['- Propellers angles : ',...
                    num2str(beta_prop_deg(2),3),...
                    ' deg (outside prop.), ', num2str(0.0,3),...
                    ' deg (inside prop.) (in floes)'])
                disp(['                      ',...
                    num2str(beta_prop_deg(1),3),...
                    ' deg (outside prop.), ', num2str(0.0,3),...
                    ' deg (inside prop.) (in level ice)'])
            case 3, % 2 pods, only the pod inside the turn rotates
                disp(['- Propellers angles : ', num2str(0.0,3),...
                    ' deg (outside prop.), ',...
                    num2str(beta_prop_deg(2),3),...
                    ' deg (inside prop.) (in floes)'])
                disp(['                      ',...
                    num2str(beta_prop_deg(1),3),...
                    ' deg (outside prop.), ', num2str(0.0,3),...
                    ' deg (inside prop.) (in level ice)'])
        end;
    end;
  if L_area < 2*R_turning*1.5,
    disp(['WARNING : L_area is quite small. ',...
          'A sector management technique might be more adapted.'])
  end;
else
  disp(['SIMULATION ABORTED : Insufficient Icebreaking power. ',...
        'Unable to manage the ice.'])
    disp(['With the given vessel and the requested target floe size, ',...
          'it is only possible'])
    disp(['to manage ', num2str(W_area_max, 3),...
          'm of the requested width of ', num2str(W_area_m, 3),...
          'm. (', num2str(100*W_area_max/W_area_m, 3), '%).'])
  disp(' Possible solutions :')
    disp(['- increase icebreaking power up to at least ',...
          num2str(P_D/1e6, 3), ' MW '])
```

```
    disp( '- use more than one management vessel')
      if P_D_lim_floe <= P_D_max_MW*1e6,
          disp(['- increase target floe size above ',...
                num2str(d_floe_target_min , 3),' m.'])
      end;
      error('SIMULATION ABORTED');
end;


%% TRAJECTORY CALCULATION

V_lin_nodrift = V_lin_ice(1);
V_lin_updrift = V_lin_ice(2);
V_lin_downdrift = V_lin_ice(3);

e_w = ((N_pass-1)*(d_floe_target + W_channel)-W_traj)/2;
traj_init = [-W_traj/2-e_w; -L_area/2+R_turning];
dist_0 = R_excl+L_area/2+W_channel/2; %Distance of the trajectory centre
                                      %to the structure centre

%Rotation matrices
Rot_psi = [cos(Psi_struct_rad) -sin(Psi_struct_rad);...
    sin(Psi_struct_rad)  cos(Psi_struct_rad)];            %for Psi_struct
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];          %for alpha_drift

l_0_up   = ceil((L_area-2*R_turning)/V_lin_updrift/dt);
l_0_down = ceil((L_area-2*R_turning)/V_lin_downdrift/dt);
l_h      = ceil((W_traj-2*R_turning+2*e_w)/V_lin_nodrift/dt);
l_180    = ceil(pi*R_turning/V_turning/dt);
l_90     = ceil(0.5*pi*R_turning/V_turning/dt);

n=1;
N_pass_c=1;

%%%% if SINGLE pass and EVEN N_pass %%%%
if (~double_pass) && (~mod(N_pass,2)),
    traj_0_t = zeros(2, N_pass*l_180 + 0.5*N_pass*l_0_up...
                                     + 0.5*N_pass*l_0_down + l_h + 2*l_90);
    traj_0_t(:,1) = traj_init;

    while N_pass_c < N_pass,
        % straight line, downdrift
        end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
        while traj_0_t(2,n) < end_str_line,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
        end;
        ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
        n = max(n,1); %     n =max(n-1,2);

        % turning 180deg
        traj_0_t(:,n:(n+l_180-1)) = ...
                (traj_0_t(:,n)-[0;ex_1])*ones(1,l_180)...
                + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                  R_turning*sin(pi:-(pi/(l_180-1)):0)];
        n=n+l_180;

        % straight line, up-drift
        n = n-1;
        end_str_line_2 = traj_0_t(2,n) - max((L_area-2*R_turning),0);
        while traj_0_t(2,n) > end_str_line_2,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
        end;

        ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance
        N_pass_c = N_pass_c+2;

        if N_pass_c < N_pass,
            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) = (traj_0_t(:,n)-[0;ex_2])...
                *ones(1,l_180)...
                + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                   R_turning*sin(-pi:(pi/(l_180-1)):0)];
            n=n+l_180-1;
        end;

    end;

    %turning 90deg
    traj_0_t(:,n:(n+l_90-1)) =  (traj_0_t(:,n)-[0;ex_2])*ones(1,l_90)...
        + [R_turning*(cos(0:(-pi/2/(l_90-1)):(-pi/2))-1);...
           R_turning*sin(0:(-pi/2/(l_90-1)):(-pi/2))];
    n=n+l_90-1;

    % perpendicular straight line
    %         n =n-1;
    end_str_line_h = traj_0_t(1,n) - W_traj+2*R_turning-2*e_w;
    while traj_0_t(1,n) > end_str_line_h,
        n =n+1;
        traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
    end;
    ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

    %turning 90deg
    traj_0_t(:,n:(n+l_90)) =  (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90+1)...
        + [R_turning*(cos((-pi/2):(-pi/2/(l_90)):-pi));...
           R_turning*sin((-pi/2):(-pi/2/(l_90)):-pi)+1)];
    N_pass_c = N_pass_c+1;
    n=n+l_90;
    traj_0_t(:,(n+1):end)=[];

    % Trajectory is repeated to have 2 loops, as it is when N_pass is odd
    traj_0_t = [traj_0_t traj_0_t(:, 2:end)];

%%%% if SINGLE pass and ODD N_pass %%%%
elseif (~double_pass) && (mod(N_pass,2)),
    traj_0_t = zeros(2, 2*N_pass*l_180 + N_pass*l_0_up...
                        + N_pass*l_0_down + 2*l_h+4*l_90);
    traj_0_t(:,1) = traj_init;

    % 1st unclosed loop
    while N_pass_c < N_pass,
        % straight line, downdrift
        end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
        while traj_0_t(2,n) < end_str_line,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
        end;
        ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
        n = max(n,1); %     n =max(n-1,2);

        % turning 180deg
        traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_1])...
            *ones(1,l_180)...
            + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
               R_turning*sin(pi:-(pi/(l_180-1)):0)];
        N_pass_c = N_pass_c+1;
        n=n+l_180;

        if N_pass_c < N_pass,

            % straight line, up-drift
            n = n-1;
            end_str_line_2 = traj_0_t(2,n) - max((L_area-2*R_turning),0);
            while traj_0_t(2,n) > end_str_line_2,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
            end;
            ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_2])...
                *ones(1,l_180)...
                + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                   R_turning*sin(-pi:(pi/(l_180-1)):0)];
            N_pass_c = N_pass_c+1;
            n=n+l_180-1;
        end;

    end;

    %turning 90deg
    traj_0_t(:,n:(n+l_90-1)) = ...
                (traj_0_t(:,n)-[0;ex_2])*ones(1,l_90)...
               + [R_turning*(cos(0:(-pi/2/(l_90-1)):(-pi/2))-1);...
                  R_turning*sin(0:(-pi/2/(l_90-1)):(-pi/2))];
    n=n+l_90-1;
    n_90_save=n;

    % perpendicular straight line
    end_str_line_h = traj_0_t(1,n)...
                     - ((N_pass1-1)*2*R_turning-3*R_turning);
```

```
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90)) = (traj_0_t(:,n)...
                        -[ex_h;0])*ones(1,l_90+1)...
                       + [R_turning*(cos((-pi/2):(-pi/2/(l_90)):-pi));...
                          R_turning*(sin((-pi/2):(-pi/2/(l_90)):-pi)+1)];
        N_pass_c = N_pass_c+1;
        n=n+l_90;
        traj_0_t(:,(n+1):end)=[];

        % We then copy the same loop, with same y orientation
        traj_0_t = [traj_0_t(1,:),...
                traj_0_t(1,2:n_90_save)-traj_0_t(1, 2)+traj_0_t(1,end);...
                traj_0_t(2,:),...
                traj_0_t(2, 2:n_90_save)];
        n = length(traj_0_t);

        % perpendicular straight line
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90)...
                       + [R_turning*(cos((pi/2):(pi/2/(l_90-1)):pi));...
                          -R_turning*(sin((pi/2):(pi/2/(l_90-1)):pi)-1)];
        n=n+l_90;

    else %ODD N_pass1
        while N_pass_c < N_pass1,
            % straight line, downdrift
            end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
            while traj_0_t(2,n) < end_str_line,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
            end;
            ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
            n = max(n,1);

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_1])...
                *ones(1,l_180)...
                + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                R_turning*sin(pi:-(pi/(l_180-1)):0)];
            N_pass_c = N_pass_c+1;
            n=n+l_180;

            if N_pass_c < N_pass1,
                % straight line, up-drift
                n = n-1;
                end_str_line_2 = traj_0_t(2,n)...
                                 -max((L_area-2*R_turning),0);
                while traj_0_t(2,n) > end_str_line_2,
                    n =n+1;
                    traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
                end;
                ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance

                % turning 180deg
                traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_2])...
                    *ones(1,l_180)...
                    + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                       R_turning*sin(-pi:(pi/(l_180-1)):0)];

                N_pass_c = N_pass_c+1;
                n=n+l_180-1;
            end;
        end;

        % End of 1st unclosed loop
        % straight line, downdrift
        end_str_line_1 = traj_0_t(2,n) + max((L_area-2*R_turning),0);
        while traj_0_t(2,n) < end_str_line_1,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
        end;
        ex_1 = traj_0_t(2,n)-end_str_line; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) =  (traj_0_t(:,n)-[0;ex_1])...
            *ones(1,l_90)...
            + [R_turning*(cos(0:(pi/2/(l_90-1)):(pi/2))-1);...
            R_turning*sin(0:(pi/2/(l_90-1)):(pi/2))];
        n=n+l_90-1;
        n_90_save=n;

        % perpendicular straight line
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1-1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance


        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) = (traj_0_t(:,n)...
                        -[ex_h;0])*ones(1,l_90)...
                       + [R_turning*(cos((pi/2):(pi/2/(l_90-1)):pi));...
                          R_turning*(sin((pi/2):(pi/2/(l_90-1)):pi)-1)];
        n=n+l_90;

        traj_0_t(:,n:end) = [];

        % We then copy the same loop, with reverse y orientation
        traj_0_t = [traj_0_t(1,:),...
                traj_0_t(1,2:n_90_save)-traj_0_t(1,2)+traj_0_t(1,end);...
                traj_0_t(2,:),...
               -traj_0_t(2, 2:n_90_save)];

        n = length(traj_0_t);
        % perpendicular straight line
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90+1)...
                       + [R_turning*(cos((pi/2):(pi/2/(l_90-1)):pi));...
                          -R_turning*(sin((pi/2):(pi/2/(l_90-1)):pi)-1)];
        n=n+l_90;
    end;


%%%% if DOUBLE pass and ODD N_pass %%%%
elseif (double_pass) && (mod(N_pass,2)),
    N_pass1 = ceil(N_pass/2);
    N_pass2 = N_pass - N_pass1;


    traj_0_t = zeros(2, N_pass*l_180 + 0.5*N_pass*l_0_up...
                                     + 0.5*N_pass*l_0_down + l_h + 2*l_90);
    traj_0_t(:,1) = traj_init;


    if ~(mod(N_pass1,2)), % EVEN N_pass1
        while N_pass_c < N_pass1,
            % straight line, downdrift
            end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
            while traj_0_t(2,n) < end_str_line,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
            end;
            ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
            n = max(n,1); %     n =max(n-1,2);

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1))=(traj_0_t(:,n)...
                            - [0;ex_1])*ones(1,l_180)...
                           + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                              R_turning*sin(pi:-(pi/(l_180-1)):0)];
            n=n+l_180;

            % straight line, up-drift
            n = n-1;
            end_str_line_2 = traj_0_t(2,n) - max((L_area-2*R_turning),0);
            while traj_0_t(2,n) > end_str_line_2,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
            end;

            ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance
            N_pass_c = N_pass_c+2;

            if N_pass_c < N_pass1,
                % turning 180deg
                traj_0_t(:,n:(n+l_180-1)) = (traj_0_t(:,n)-[0;ex_2])...
                                *ones(1,l_180)...
                               + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                                  R_turning*sin(-pi:(pi/(l_180-1)):0)];
                n=n+l_180-1;
            end;

        end;


        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) =  (traj_0_t(:,n)...
                        - [0;ex_2])*ones(1,l_90)...
                       + [R_turning*(cos(0:(-pi/2/(l_90-1)):(-pi/2))-1);...
                          R_turning*sin(0:(-pi/2/(l_90-1)):(-pi/2))];
        n=n+l_90-1;

        % perpendicular straight line
        %         n =n-1;
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1-1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90+1)...
                       + [R_turning*(cos((-pi/2):(-pi/2/(l_90)):-pi));...
                          R_turning*(sin((-pi/2):(-pi/2/(l_90)):-pi)+1)];
        n=n+l_90;
        traj_0_t(:,(n+1):end)=[];


        N_pass_c = 1;
        while N_pass_c < N_pass2,
            % straight line, downdrift
            end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
            while traj_0_t(2,n) < end_str_line,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
            end;
            ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
            n = max(n,1);

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_1])...
                *ones(1,l_180)...
                + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                R_turning*sin(pi:-(pi/(l_180-1)):0)];
            N_pass_c = N_pass_c+1;
            n=n+l_180;

            if N_pass_c < N_pass2,

                % straight line, up-drift
                n = n-1;
                end_str_line_2 = traj_0_t(2,n)...
                                 - max((L_area-2*R_turning),0);
                while traj_0_t(2,n) > end_str_line_2,
                    n =n+1;
                    traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
                end;
                ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance

                % turning 180deg
                traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_2])...
                    *ones(1,l_180)...
                    + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                    R_turning*sin(-pi:(pi/(l_180-1)):0)];

                N_pass_c = N_pass_c+1;
                n=n+l_180-1;
            end;
```

```
        end;

        % End of 1st unclosed loop
        % straight line, downdrift
        end_str_line_1 = traj_0_t(2,n) + max((L_area-2*R_turning),0);
        while traj_0_t(2,n) < end_str_line_1,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
        end;
        ex_1 = traj_0_t(2,n)-end_str_line; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) =  (traj_0_t(:,n)-[0;ex_1])...
            *ones(1,l_90)...
            + [R_turning*(cos(0:(pi/2/(l_90-1)):(pi/2))-1);...
            R_turning*sin(0:(pi/2/(l_90-1)):(pi/2))];
        n=n+l_90-1;
        n_90_save=n;

        % perpendicular straight line
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass2)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance


        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90)...
                       + [R_turning*(cos((pi/2):(pi/2/(l_90-1)):pi));...
                          R_turning*(sin((pi/2):(pi/2/(l_90-1)):pi)-1)];
        n=n+l_90;

        traj_0_t(:,n:end) = [];

        % We then copy the same loop, with reverse y orientation, so that
        % the trajectory is now closed
        traj_0_t = [traj_0_t(1,:),...
                    traj_0_t(1, 2:end)-traj_0_t(1, 2)+traj_0_t(1,end);...
                    traj_0_t(2,:),...
                   -traj_0_t(2, 2:end)];


    else %ODD N_pass1
        while N_pass_c < N_pass1,
            % straight line, downdrift
            end_str_line = traj_0_t(2,n) + max((L_area-2*R_turning),0);
            while traj_0_t(2,n) < end_str_line,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
            end;
            ex_1 = traj_0_t(2,n)-end_str_line; % exceedance
            n = max(n,1);

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) =  (traj_0_t(:,n)-[0;ex_1])...
                *ones(1,l_180)...
                + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                R_turning*sin(pi:-(pi/(l_180-1)):0)];
            N_pass_c = N_pass_c+1;
            n=n+l_180;

            if N_pass_c < N_pass1,

                % straight line, up-drift
                n = n-1;
                end_str_line_2 = traj_0_t(2,n)...
                                 - max((L_area-2*R_turning),0);
                while traj_0_t(2,n) > end_str_line_2,
                    n =n+1;
                    traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
                end;
                ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance

                % turning 180deg
                traj_0_t(:,n:(n+l_180-1)) = (traj_0_t(:,n)-[0;ex_2])...
                            *ones(1,l_180)...
                          + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                            R_turning*sin(-pi:(pi/(l_180-1)):0)];

                N_pass_c = N_pass_c+1;
                n=n+l_180-1;
            end;
        end;

        % End of 1st unclosed loop
        % straight line, downdrift
        end_str_line_1 = traj_0_t(2,n) + max((L_area-2*R_turning),0);
        while traj_0_t(2,n) < end_str_line_1,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
        end;
        ex_1 = traj_0_t(2,n)-end_str_line; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) =  (traj_0_t(:,n)-[0;ex_1])...
            *ones(1,l_90)...
            + [R_turning*(cos(0:(pi/2/(l_90-1)):(pi/2))-1);...
            R_turning*sin(0:(pi/2/(l_90-1)):(pi/2))];
        n=n+l_90-1;

        % perpendicular straight line
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1-1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance


        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90)...
                       + [R_turning*(cos((pi/2):(pi/2/(l_90-1)):pi));...
                          R_turning*(sin((pi/2):(pi/2/(l_90-1)):pi)-1)];
        n=n+l_90;

        N_pass_c = 1;

        while N_pass_c < N_pass2,
            % straight line, up-drift
            n = n-1;
            end_str_line_2 = traj_0_t(2,n) - max((L_area-2*R_turning),0);
            while traj_0_t(2,n) > end_str_line_2,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;-1]*V_lin_updrift*dt;
            end;
            ex_2 = traj_0_t(2,n)-end_str_line_2; % exceedance

            % turning 180deg
            traj_0_t(:,n:(n+l_180-1)) = (traj_0_t(:,n)-[0;ex_2])...
                *ones(1,l_180)...
                + [R_turning*(cos(-pi:(pi/(l_180-1)):0)+1);...
                R_turning*sin(-pi:(pi/(l_180-1)):0)];
            n=n+l_180-1;


            % straight line, downdrift
            end_str_line_1 = traj_0_t(2,n) + max((L_area-2*R_turning),0);
            while traj_0_t(2,n) < end_str_line_1,
                n =n+1;
                traj_0_t(:,n)=traj_0_t(:,n-1)+[0;1]*V_lin_downdrift*dt;
            end;
            ex_1 = traj_0_t(2,n)-end_str_line; % exceedance


            N_pass_c = N_pass_c+2;

            if N_pass_c < N_pass2,
                % turning 180deg
                traj_0_t(:,n:(n+l_180-1))=(traj_0_t(:,n)...
                            - [0;ex_1])*ones(1,l_180)...
                           + [R_turning*(cos(pi:-(pi/(l_180-1)):0)+1);...
                              R_turning*sin(pi:-(pi/(l_180-1)):0)];
                n=n+l_180;
            end;

        end;


        %turning 90deg
        traj_0_t(:,n:(n+l_90-1)) = ...
                        (traj_0_t(:,n)-[0;ex_1])*ones(1,l_90)...
                       + [ R_turning*(cos(0:(-pi/2/(l_90-1)):(-pi/2))-1);...
                         -R_turning*sin(0:(-pi/2/(l_90-1)):(-pi/2))];
        n=n+l_90-1;
        n_90_save=n;

        % perpendicular straight line
        %       n =n-1;
        end_str_line_h = traj_0_t(1,n)...
                         - ((N_pass1-1)*2*R_turning-3*R_turning);
        while traj_0_t(1,n) > end_str_line_h,
            n =n+1;
            traj_0_t(:,n)=traj_0_t(:,n-1)+[-1;0]*V_lin_nodrift*dt;
        end;
        ex_h = traj_0_t(1,n)-end_str_line_h; % exceedance

        %turning 90deg
        traj_0_t(:,n:(n+l_90)) = ...
                        (traj_0_t(:,n)-[ex_h;0])*ones(1,l_90+1)...
                       + [ R_turning*(cos((-pi/2):(-pi/2/(l_90)):-pi));...
                          -R_turning*(sin((-pi/2):(-pi/2/(l_90)):-pi)+1)];
        N_pass_c = N_pass_c+1;
        n=n+l_90;
        traj_0_t(:,(n+1):end)=[];

        % We then copy the same loop, with reverse y orientation
        traj_0_t = [traj_0_t(1,:),...
                    traj_0_t(1, 2:end)-traj_0_t(1, 2)+traj_0_t(1,end);...
                    traj_0_t(2,:),...
                   -traj_0_t(2, 2:end)];
    end;

end;


traj_t = zeros(size(traj_0_t));

%% VIEW

% NB : the structure centre is in (0,0)

% Structure and critical area
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    draw_struct = [(D_struct/2)*cos(0:(pi/50):(2*pi));...
        (D_struct/2)*sin(0:(pi/50):(2*pi))];
    draw_crit   = [(D_struct/2+t_cr)*cos(0:(pi/50):(2*pi));...
        (D_struct/2+t_cr)*sin(0:(pi/50):(2*pi))];

elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    draw_struct_0 = [-B_struct/2, -B_struct/2, B_struct/2, B_struct/2,...
                     -B_struct/2;...
                     -L_struct/2,  L_struct/2, L_struct/2,-L_struct/2,...
                     -L_struct/2,];

    draw_crit_0   = [-B_struct/2+t_cr*sin(-pi:(pi/50):(-pi/2)),...
        -B_struct/2+t_cr*sin((-pi/2):(pi/50):0),...
        B_struct/2+t_cr*sin(0:(pi/50):(pi/2)),...
        B_struct/2+t_cr*sin((pi/2):(pi/50):pi),...
        -B_struct/2;...
        -L_struct/2+t_cr*cos(-pi:(pi/50):(-pi/2)),...
        L_struct/2+t_cr*cos((-pi/2):(pi/50):0),...
        L_struct/2+t_cr*cos(0:(pi/50):(pi/2)),...
        -L_struct/2+t_cr*cos((pi/2):(pi/50):pi),...
        -L_struct/2-t_cr];

    draw_struct = zeros(2,5);
    draw_crit   = zeros(2,length(draw_crit_0(1,:)));
    for k=1:5,
        draw_struct(:,k) = draw_struct_0(:,k)'*Rot_psi;
    end;
    for k=1:length(draw_crit_0(1,:)),
        draw_crit(:,k) = draw_crit_0(:,k)'*Rot_psi;
    end;
end;

% Exclusion zone
draw_excl_zone = [R_excl*cos(0:(pi/50):(2*pi));...
                  R_excl*sin(0:(pi/50):(2*pi))];

% Channel
draw_channel_0= [-W_area_m/2,  -W_area_m/2,   W_area_m/2,  W_area_m/2;...
                 -1.5*(R_excl+L_area), 1.5*(R_excl+L_area),...
                  1.5*(R_excl+L_area), -1.5*(R_excl+L_area)];

% Ice management area
draw_area_0 = [-W_area_m/2,  W_area_m/2, W_area_m/2,-W_area_m/2,...
               -W_area_m/2;...
               -L_area/2-W_channel/2, -L_area/2-W_channel/2,...
                L_area/2+W_channel/2,  L_area/2+W_channel/2,...
               -L_area/2-W_channel/2];
```

```
draw_channel = zeros(2,4);
draw_area    = zeros(2,4);
for k=1:4,
    draw_channel(:,k) = draw_channel_0(:,k)'*Rot_alpha;
end;
for k=1:5,
    draw_area(:,k)    = draw_area_0(:,k)'*Rot_alpha;
end;

draw_area(1,:) = draw_area(1,:) - dist_0*sin(alpha_drift_rad);
draw_area(2,:) = draw_area(2,:) - dist_0*cos(alpha_drift_rad);

% Representation
n_traj      = 1;
frame_freq  = -1;%Value for representation only in Simulation_calc funct.
draw_labels = true;
mov         = {false};

if double_pass,
    N_pass_r = ceil(N_pass/2)+1;
else
    N_pass_r = N_pass;
end;
X_lims_add = [-1.2*(N_pass_r-1)*R_turning*cos(alpha_drift_rad);...
               1.2*(N_pass_r-1)*R_turning*cos(alpha_drift_rad)];
Y_lims_add = [-1.2*(N_pass_r-1)*R_turning*sin(alpha_drift_rad);...
               1.2*(N_pass_r-1)*R_turning*sin(alpha_drift_rad)];

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area_m, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
end;


%% SIMULATION
n_traj      = 1.50;
frame_freq  = 20;
draw_labels = true;
if movie_rec,
    mov = {true; 'IM_linear_single_pass'; 15};
else
    mov = {false};
end;

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area_m, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
end;

%% FLOE SIZE ANALYSIS
t_cycle   = dt*length(traj_t);
fig_local = false;

%Specific simulation for floe analysis (without display)
n_traj      = 10;
frame_freq  = 0;
draw_labels = false;
mov         = {false};
X_lims_add  = [];
Y_lims_add  = [];
[ice_ch_1, ice_ch_2] = Simulation_calc(traj_0_t, n_traj, frame_freq,...
                draw_labels, mov, W_channel, L_area, W_area_m, X_lims_add,...
                Y_lims_add, draw_struct, draw_crit, draw_excl_zone,...
                draw_channel_0, draw_area, alpha_drift, dist_0);

% Floe analysis
Floe_size_analysis_linear(ice_ch_1, ice_ch_2, N_pass, W_channel,...
                                                t_cycle, fig_local, fig_main);
```

### *D.7.2 L_area_fct.m*

```
function [L_area, P_D, V_lin_ice, R_turning, V_turning, beta_prop_deg,...
          beta_drift_deg, P_D_turning] = L_area_fct(V_drift, ex_l_area,...
          W_channel, d_floe_target, w_traj, prop_mode, advance_mode,...
          fig_local)
%
% Compute the length of the area to be managed, for the linear management
% technique.
%
% INPUTS :
% V_drift     [m/s]   Ice drift velocity
% ex_l_area    [-]    L_area_90 (i.e. L_area with 90% of P_D) has to be
%                       less than ex_l_area*L_area
% W_channel    [m]    Icebreaker channel width
% d_floe_target [m]   Target floe size
% w_traj       [m]    Width of the vessel trajectory
% prop_mode    [-]    Propeller(s) rotating mode :
%                       1: All propellers are rotating
%                       2: 2 pods, only the pod outside the turn rotates
%                       3: 2 pods, only the pod inside the turn rotates
% turning_mode  [-]   Type of turning :
%                       1: Keeping same power as in linear motion, V
%                          decreases
%                       2: Keeping same velocity as in linear motion,
%                          P_D increases
% fig_local     [Bool] Activate/Deactivate local figures plotting
%                      (V_turning vs P_D, Thrust and resistance diagrams)
%
% OUTPUTS:
% L_area          [m]    Length of area to be managed
% P_D       [W]    Required delivered power (in level ice, and in
%                      floes if applicable)
% V_lin_ice       [m/s]  Linear motion velocities
%                        (no-drift,up-drift,down-drift)
% R_turning       [m]    Turning radius
% V_turning       [m/s]  Velocity of the ship's centre of gravity in turns
% beta_prop_deg   [deg]  Propeller(s) orientation angle (in level ice, and
%                        in floes if applicable)
% beta_drift_deg  [deg]  Ship's drift angle
% P_D_turning     [W]    Required turning delivered power (in level ice,
%                        and in floes if applicable)
%

global Ship_data
global Structure_data

% READING SHIP PARAMETERS
B          = Ship_data{3};    % Ship's breadth
P_D_max_MW = Ship_data{14}(1);% Ship's max delivered power (! in MW)

% READING STRUCTURE PARAMETERS
double_pass = Structure_data{7};% True if double pass management technique

% Number of passes of the ice management trajectory (linear method)
N_pass = ceil(w_traj/(W_channel+d_floe_target))+1;


V_drift_calc   = [0, V_drift, -V_drift];

% Required turning radius
R_turning = (d_floe_target+W_channel)/2;
if double_pass,
    R_turning = 2*R_turning;

    d_floe_target = Structure_data{6};% Target floe size
    k_channel   = Ship_data{18};     % Channel width to ship's beam ratio
    W_channel   = B*k_channel;       % Icebreaker channel width
    d_floe_pass1  = 2*d_floe_target;
    C_ice_pass1   = (d_floe_pass1^2)...
                    /(d_floe_pass1^2 + d_floe_pass1*W_channel);
end;



% TURNING ESTIMATION
disp('- Turning estimation')
turn_sign = +1; % +1 for turn to port (arbitrary choice)
P_D_vect = [(1e6*P_D_max_MW/5):(1e6*P_D_max_MW/10):(1e6*P_D_max_MW),...
            (1e6*(4/3)*P_D_max_MW):(1e6*P_D_max_MW/3):(1e6*3.5*P_D_max_MW)];
P_D_fl_vect = P_D_vect;
V_turning_vect      = zeros(1, length(P_D_vect));
beta_prop_deg_vect  = zeros(1, length(P_D_vect));
beta_drift_deg_vect = zeros(1, length(P_D_vect));
if double_pass,
    V_turning_fl_vect      = zeros(1, length(P_D_vect));
    beta_prop_deg_fl_vect  = zeros(1, length(P_D_vect));
    beta_drift_deg_fl_vect = zeros(1, length(P_D_vect));
end;
for i = 1:length(P_D_vect),
    disp([num2str(i), '/', num2str(length(P_D_vect))])
    %Level ice
    [V_turning_vect(i), beta_prop_deg_vect(i), beta_drift_deg_vect(i)]=...
                    Calc_circle_P_D(R_turning, P_D_vect(i), turn_sign,...
                    prop_mode, false, fig_local);
    if double_pass,
        %Floes
        [V_turning_fl_vect(i), beta_prop_deg_fl_vect(i),...
                               beta_drift_deg_fl_vect(i)] =...
            Calc_circle_P_D(R_turning, P_D_fl_vect(i), turn_sign,...
            prop_mode, true, fig_local);
    end;
end;
i=1;
while i <= length(V_turning_vect), % Removing the NaN results, level ice
    if isnan(V_turning_vect(i)),
        V_turning_vect(i)       = [];
        beta_prop_deg_vect(i)   = [];
        beta_drift_deg_vect(i)  = [];
        P_D_vect(i)             = [];
    else
        i = i+ 1;
    end;
end;
if double_pass,
    while i <= length(V_turning_fl_vect),% Removing the NaN results, floes
        if isnan(V_turning_fl_vect(i)),
            V_turning_fl_vect(i)       = [];
            beta_prop_deg_fl_vect(i)   = [];
            beta_drift_deg_fl_vect(i)  = [];
            P_D_fl_vect(i)             = [];
        else
            i = i+ 1;
        end;
    end;
end;

if fig_local,
    figure,
    plot(P_D_vect/1e6, V_turning_vect)
    if double_pass,
        hold on,
        plot(P_D_vect/1e6, V_turning_fl_vect, 'r')
        legend('Level ice', 'Floes', 'Location', 'SouthEast')
    end;
    xlabel('P_D [MW]', 'Fontsize', 12)
    ylabel('V turning [m/s]', 'Fontsize', 12)
end;


if advance_mode == 2,
    % Calculation of turning at maximum delivered power, often required,
    % as turning at same velocity as in linear motion is often too
    % demanding in delivered power
    [V_turning_max_P_D_lvl, beta_prop_deg_max_P_D,...
                beta_drift_deg_max_P_D] = ...
                Calc_circle_P_D(R_turning, P_D_max_MW*1e6, turn_sign,...
                prop_mode, false, fig_local);
    if double_pass,
        [V_turning_max_P_D_fl, beta_prop_deg_max_P_D_fl,...
                beta_drift_deg_max_P_D_fl] = ...
                Calc_circle_P_D(R_turning, P_D_max_MW*1e6, turn_sign,...
                prop_mode, true, fig_local);
    end;
end;


%ITERATING FOR L_area
disp('- Iterating for L_area')
disp_eq = false; % Disable equilibrium velocity display
% Initial values
P_D        = P_D_max_MW*1e6;
L_area     = 1;
L_area_90 = 0;
N_it = 1;
N_it_max = 100; %Maximum number of iterations
err = 1;
while (abs(err) > 1e-2) && (N_it < N_it_max),
    Power_OK = false; % To manage the problem when P_D is too low, so that
                      % V_lvl_ice = 0 and then L_area = NaN.
    while ~Power_OK,
        %%%%%% L_area %%%%%%
        % LINEAR MOTION IN LEVEL ICE
        V_lvl_ice = zeros(1,3);
        for k=1:3,
            V_lvl_ice(k) = Equil_level_ice(P_D, V_drift_calc(k),...
                                                    disp_eq, fig_local);
        end;
        V_lvl_nodrift = V_lvl_ice(1);
        V_lvl_updrift = V_lvl_ice(2);
        V_lvl_downdrift = V_lvl_ice(3);

        % TURNING IN LEVEL ICE
        % Velocity, propeller(s) angle and drift angle
        if advance_mode == 1, %Turning at same power as linear motion
            V_lvl_turning   = interp1(P_D_vect, V_turning_vect, P_D);
            P_D_turning = P_D;
            if isnan(V_lvl_turning),
                %P_D is out of the range already calculated
                turn_floes = false;
                [V_lvl_turning, beta_prop_deg, beta_drift_deg] =...
                      Calc_circle_P_D(R_turning, P_D_turning, turn_sign,...
                      prop_mode, turn_floes, fig_local);
                if isnan(V_lvl_turning),
                    disp(['ERROR : Unable to find a turning velocity',...
                          ' for the requested power level.'])
                    error('SIMULATION ABORTED');
                end;
            else
                beta_prop_deg  =interp1(P_D_vect,beta_prop_deg_vect, P_D);
                beta_drift_deg =interp1(P_D_vect,beta_drift_deg_vect,P_D);
            end;
        elseif advance_mode == 2,%Turning at same velocity as lin. motion
            V_lvl_turning   = V_lvl_nodrift;
            P_D_turning = interp1(V_turning_vect,P_D_vect,V_lvl_turning);
            if isnan(P_D_turning),
                %disp('out of range - calculation')
                %V_lvl_turning is out of the range already calculated
                turn_floes = false;
                [P_D_turning, beta_prop_deg, beta_drift_deg] =...
                      Calc_circle_V(R_turning, V_lvl_turning, turn_sign,...
                      prop_mode, turn_floes, fig_local);
                if isnan(P_D_turning),
                    disp(['ERROR : Unable to find a delivered power ',...
                          ' for the requested turning velocity.'])
                    error('SIMULATION ABORTED');
                end;
            else
                beta_prop_deg = interp1(V_turning_vect,...
                                beta_prop_deg_vect, V_lvl_turning);
                beta_drift_deg = interp1(V_turning_vect,...
                                beta_drift_deg_vect, V_lvl_turning);
            end;
            if P_D_turning > (P_D_max_MW*1e6),
                P_D_turning    = P_D_max_MW*1e6;% turning at maximum power
                V_lvl_turning  = V_turning_max_P_D_lvl;
                beta_prop_deg  = beta_prop_deg_max_P_D;
                beta_drift_deg = beta_drift_deg_max_P_D;
                if isnan(V_lvl_turning),
                    disp(['ERROR : Unable to find a turning velocity',...
                          ' for the requested power level.'])
                    error('SIMULATION ABORTED');
                end;
            end;
        else
            disp('Unknown turning mode.')
            error('SIMULATION ABORTED')
        end;

        % ADVANCE IN FLOES (if applicable)
        if double_pass,
            V_fl_ice = zeros(1,3);
            if advance_mode == 1,     %Same power as level ice
                % LINEAR MOTION IN FLOES
                P_D_fl   = P_D;
                for k=1:3,
                    V_fl_ice(k) = Equil_floes(P_D_fl, V_drift_calc(k),...
                                  d_floe_pass1, C_ice_pass1, disp_eq,...
                                  fig_local);
                end;
                % TURNING IN FLOES
                V_fl_turning   = interp1(P_D_fl_vect,V_turning_fl_vect,...
                                                        P_D_fl);
                P_D_fl_turning = P_D_fl;
                if isnan(V_fl_turning),
                    %P_D_fl is out of the range already calculated
                    turn_floes = true;
                    [V_fl_turning, beta_prop_deg_fl,...
```

```
                beta_drift_deg_fl] = ...
                Calc_circle_P_D(R_turning, P_D_fl_turning,...
                turn_sign, prop_mode, turn_floes, fig_local);
            if isnan(V_fl_turning),
                disp(['ERROR : Unable to find a turning ',...
                      'velocity in floes for the requested ',...
                      'power level.'])
                error('SIMULATION ABORTED');
            end;
        else
            beta_prop_deg_fl  = interp1(P_D_fl_vect,...
                                    beta_prop_deg_fl_vect, P_D_fl);
            beta_drift_deg_fl = interp1(P_D_fl_vect,...
                                    beta_drift_deg_fl_vect,P_D_fl);
        end;

    elseif advance_mode == 2, %Same velocity as linear motion
        % LINEAR MOTION IN FLOES
        V_fl_ice = V_lvl_ice;
        FCT_fl_P_D = @(P_D_fl) V_fl_ice(1)-Equil_floes(P_D_fl,...
                    V_drift_calc(k), d_floe_pass1,C_ice_pass1,...
                    disp_eq, fig_local);
        P_D_fl   = fzero(FCT_fl_P_D, [P_D/100 P_D]);


        % TURNING IN FLOES
        V_fl_turning   = V_lvl_nodrift;
        P_D_fl_turning = interp1(V_turning_fl_vect,P_D_fl_vect,...
                         V_fl_turning);
        if isnan(P_D_fl_turning),
            %disp('out of range - calculation')
            %V_fl_turning is out of the range already calculated
            turn_floes = true;
            [P_D_fl_turning, beta_prop_deg_fl,...
                beta_drift_deg_fl] = ...
            Calc_circle_V(R_turning,V_fl_turning,turn_sign,...
            prop_mode, turn_floes, fig_local);
            if isnan(P_D_fl_turning),
                disp(['ERROR : Unable to find a delivered ',...
                      'power for the requested turning ',...
                      'velocity in floes.'])
                error('SIMULATION ABORTED');
            end;
        else
            beta_prop_deg_fl = interp1(V_turning_fl_vect,...
                beta_prop_deg_fl_vect, V_fl_turning);
            beta_drift_deg_fl = interp1(V_turning_fl_vect,...
                beta_drift_deg_fl_vect, V_fl_turning);
        end;
        if P_D_fl_turning > (P_D_max_MW*1e6),
            P_D_fl_turning = P_D_max_MW*1e6;%turning at max. power
            V_fl_turning   = V_turning_max_P_D_fl;
            beta_prop_deg_fl  = beta_prop_deg_max_P_D_fl;
            beta_drift_deg_fl = beta_drift_deg_max_P_D_fl;
            if isnan(V_lvl_turning),
                disp(['ERROR : Unable to find a turning ',...
                      'velocity for the requested power level.'])
                error('SIMULATION ABORTED');
            end;
        end;
    else
        disp('Unknown turning mode.')
        error('SIMULATION ABORTED')
    end;
end;

% L_AREA
if ~double_pass,
    %single pass
    L_area = V_drift*((-N_pass*R_turning)/V_lvl_updrift...
        +(-N_pass*R_turning)/V_lvl_downdrift...
        +(w_traj-2*R_turning)/V_lvl_nodrift...
        +N_pass*pi*R_turning/V_lvl_turning)...
        /(1-V_drift*(N_pass/(2*V_lvl_updrift)...
        +N_pass/(2*V_lvl_downdrift)));

    V_lin_ice = V_lvl_ice;
    V_turning = V_lvl_turning;
else
    %double pass
    V_mean_ice       = 2./(1./V_lvl_ice + 1./V_fl_ice);
    V_mean_nodrift   = V_mean_ice(1);
    V_mean_updrift   = V_mean_ice(2);
    V_mean_downdrift = V_mean_ice(3);
    V_mean_turning   = 2/(1/V_lvl_turning + 1/V_fl_turning);

    L_area = V_drift*((-N_pass*R_turning)/V_mean_updrift...
        +(-N_pass*R_turning)/V_mean_downdrift...
        +2*(w_traj-3*R_turning)/V_mean_nodrift...
        +N_pass*pi*R_turning/V_mean_turning)...
        /(1-V_drift*(N_pass/(2*V_mean_updrift)...
        +N_pass/(2*V_mean_downdrift)));


    V_lin_ice = V_mean_ice;
    V_turning = V_mean_turning;
end;


%%%%%% L_area_90 %%%%%%
% LINEAR MOTION IN LEVEL ICE
V_lvl_ice_90 = zeros(1,3);
for k=1:3,
    V_lvl_ice_90(k) = Equil_level_ice(0.9*P_D, V_drift_calc(k),...
                disp_eq, fig_local);
end;
V_lvl_nodrift_90   = V_lvl_ice_90(1);
V_lvl_updrift_90   = V_lvl_ice_90(2);
V_lvl_downdrift_90 = V_lvl_ice_90(3);

% TURNING IN LEVEL ICE
% Velocity, propeller(s) angle and drift angle
if advance_mode == 1, %Turning at same power as linear motion
    V_lvl_turning_90 = interp1(P_D_vect, V_turning_vect, 0.9*P_D);
    if isnan(V_lvl_turning_90),
        turn_floes = false;
        V_lvl_turning_90 = Calc_circle_P_D(R_turning, 0.9*P_D,...
            turn_sign, prop_mode, turn_floes, fig_local);
        if isnan(V_lvl_turning_90),
            disp(['ERROR : Unable to find a turning velocity',...
                  ' for the requested power level.'])
            error('SIMULATION ABORTED');
        end;
    end;
elseif advance_mode == 2,%Turning at same velocity as lin. motion
    V_lvl_turning_90  = V_lvl_nodrift_90;
    P_D_turning_90 = interp1(V_turning_vect, P_D_vect,...
                             V_lvl_turning_90);
    if isnan(P_D_turning_90),
        %disp('out of range - calculation')
        %V_lvl_turning is out of the range already calculated
        turn_floes = false;
        P_D_turning_90 = Calc_circle_V(R_turning,...
                V_lvl_turning_90, turn_sign, prop_mode,...
                turn_floes, fig_local);
        if isnan(P_D_turning_90),
            disp(['ERROR : Unable to find a delivered power ',...
                  ' for the requested turning velocity.'])
            error('SIMULATION ABORTED');
        end;
    end;
    if P_D_turning_90 > (P_D_max_MW*1e6),
        V_lvl_turning_90 = V_turning_max_P_D_lvl; %turning at max.
                                                  %power
        if isnan(V_lvl_turning_90),
            disp(['ERROR : Unable to find a turning velocity',...
                  ' for the requested power level.'])
            error('SIMULATION ABORTED');
        end;
    end;
else
    disp('Unknown turning mode.')
    error('SIMULATION ABORTED')
end;

% ADVANCE IN FLOES (if applicable)
if double_pass,
    V_fl_ice_90 = zeros(1,3);
    if advance_mode == 1,     %Same power as level ice
        % LINEAR MOTION IN FLOES
        P_D_fl_90   = 0.9*P_D;
        for k=1:3,
            V_fl_ice_90(k) = Equil_floes(P_D_fl_90,...
                V_drift_calc(k), d_floe_pass1, C_ice_pass1,...
                disp_eq, fig_local);
        end;
        % TURNING IN FLOES
        V_fl_turning_90   = interp1(P_D_fl_vect,...
                                    V_turning_fl_vect, P_D_fl_90);
        P_D_turning_fl_90 = P_D_fl_90;
        if isnan(V_fl_turning_90),
            %P_D_fl is out of the range already calculated
            turn_floes = true;
            V_fl_turning_90 = Calc_circle_P_D(R_turning,...
                P_D_turning_fl_90, turn_sign, prop_mode,...
                turn_floes, fig_local);
            if isnan(V_fl_turning_90),
                disp(['ERROR : Unable to find a turning ',...
                      'velocity in floes for the requested ',...
                      'power level.'])
                error('SIMULATION ABORTED');
            end;
        end;
    elseif advance_mode == 2, %Same velocity as linear motion
        % LINEAR MOTION IN FLOES
        V_fl_ice_90 = V_lvl_ice_90;
        FCT_fl_P_D_90 = @(P_D_fl_90) V_fl_ice_90(1)...
                        - Equil_floes(P_D_fl_90, V_drift_calc(k),...
                        d_floe_pass1, C_ice_pass1, disp_eq,...
                        fig_local);
        P_D_fl_90   = fzero(FCT_fl_P_D_90, [0.9*P_D/100 0.9*P_D]);


        % TURNING IN FLOES
        V_fl_turning_90   = V_lvl_nodrift_90;
        P_D_fl_turning_90 = interp1(V_turning_fl_vect,...
                                    P_D_fl_vect, V_fl_turning_90);
        if isnan(P_D_fl_turning),
            %disp('out of range - calculation')
            %V_fl_turning is out of the range already calculated
            turn_floes = true;
            P_D_fl_turning_90 =...
                Calc_circle_V(R_turning, V_fl_turning_90,...
                turn_sign, prop_mode, turn_floes, fig_local);
            if isnan(P_D_fl_turning_90),
                disp(['ERROR : Unable to find a delivered ',...
                      'power for the requested turning ',...
                      'velocity in floes.'])
                error('SIMULATION ABORTED');
            end;
        end;
        if P_D_fl_turning_90 > (P_D_max_MW*1e6),
            P_D_fl_turning_90 = P_D_max_MW*1e6; %turning at max.
                                                %power
            V_fl_turning_90  = V_turning_max_P_D_fl;
            if isnan(V_lvl_turning),
                disp(['ERROR : Unable to find a turning ',...
                      'velocity for the requested power level.'])
                error('SIMULATION ABORTED');
            end;
        end;

    else
        disp('Unknown turning mode.')
        error('SIMULATION ABORTED')
    end;
end;

% L_AREA_90
if ~double_pass,
    %single pass
    L_area_90 = V_drift*((-N_pass*R_turning)/V_lvl_updrift_90...
        +(-N_pass*R_turning)/V_lvl_downdrift_90...
        +(w_traj-2*R_turning)/V_lvl_nodrift_90...
        +N_pass*pi*R_turning/V_lvl_turning_90)...
        /(1-V_drift*(N_pass/(2*V_lvl_updrift_90)...
        +N_pass/(2*V_lvl_downdrift_90)));
else
    %double pass
    V_mean_ice_90       = 2./(1./V_lvl_ice_90 + 1./V_fl_ice_90);
    V_mean_nodrift_90   = V_mean_ice_90(1);
    V_mean_updrift_90   = V_mean_ice_90(2);
    V_mean_downdrift_90 = V_mean_ice_90(3);
    V_mean_turning_90   = 2/...
                          (1/V_lvl_turning_90+1/V_fl_turning_90);

    L_area_90 = V_drift*((-N_pass*R_turning)/V_mean_updrift_90...
        +(-N_pass*R_turning)/V_mean_downdrift_90...
        +2*(w_traj-3*R_turning)/V_mean_nodrift_90...
        +N_pass*pi*R_turning/V_mean_turning_90)...
        /(1-V_drift*(N_pass/(2*V_mean_updrift_90)...
        +N_pass/(2*V_mean_downdrift_90)));
end;

if P_D == 0,
    P_D = 1e5;
end;

if (min(V_lvl_ice) <= 1e-5) || (L_area_90 < 0),
    P_D = 1.1*P_D;
```

```
        else
            Power_OK = true;
        end;

    end;
    err = (ex_l_area*L_area - L_area_90)/(ex_l_area*L_area);
    % New value for P_D :
    % Rem : Newton's method is not used because it appears to be unstable:
    % the first iterations often go outside the stable calculation range.
    % A "slower" convergence method is used.
    if err > 0 && N_it==1,
        P_D_nm1 = P_D;
        P_D = 0.9*P_D;
        err_nm1 = err;
    elseif err < 0 && N_it==1,
        P_D_nm1 = P_D;
        err_nm1 = err;
        P_D     = 1.1*P_D_nm1;
    elseif err < 0 && err_nm1 < 0,
        % means : both P_D and P_D_nm1 give a negative error
        P_D_nm2 = P_D_nm1;
        P_D_nm1 = P_D;
        err_nm1 = err;
        P_D     = 1.1*max(P_D_nm1,P_D_nm2);
    elseif err < 0 && err_nm1 > 0,
        % means : P_D_nm1 gives a positive error, P_D a negative error
        P_D_nm2 = P_D_nm1;
        P_D_nm1 = P_D;
        err_nm1 = err;
        P_D     = 0.5*(P_D+P_D_nm2);
    elseif err > 0 && err_nm1 < 0,
        % means : P_D_nm1 gives a negative error, P_D a positive error
        P_D_nm2 = P_D_nm1;
        P_D_nm1 = P_D;
        err_nm1 = err;
        P_D     = 0.5*(P_D+P_D_nm2);
    elseif err > 0 && err_nm1 > 0,
        % means : both P_D and P_D_nm1 give a positive error
        P_D_nm2 = P_D_nm1;
        P_D_nm1 = P_D;
        err_nm1 = err;
        P_D     = 0.9*min(P_D_nm1,P_D_nm2);
    end;

    N_it = N_it + 1;
end;

if N_it >= N_it_max,
    disp('Warning : maximum mumber of iterations has been reached.')
end;


if ~double_pass,
    %Single pass
    P_D = P_D_nm1;
else
    %Double pass
    P_D = [P_D_nm1, P_D_fl];
    P_D_turning = [P_D_turning, P_D_fl_turning];
    beta_prop_deg = [beta_prop_deg; beta_prop_deg_fl];
end;
end
```

### *D.7.3 Ice_management_sector.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                                SIMULATION OF THE                              %
%                          SECTOR ICE MANAGEMENT TECHNIQUE                      %
%                                                                               %
%                                 Quentin HISETTE                               %
%                                                                               %
%                                    2013-2014                                  %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all
close all
clc

fig_local = false;  % [Bool]  Activate/Deactivate local figures plotting
fig_main  = true;   % [Bool]  Activate/Deactivate main figures plotting
movie_rec = false;  % [Bool]  Activate/Deactivate movie recording

% READING INPUT FILES
disp('Reading input files')
Input_reading;
global Ship_data
global Structure_data
global Ice_drift_data
global Other_data

% READING STRUCTURE PARAMETERS
Ref           = Structure_data{1};    % Reference
L_struct      = Structure_data{2}(1); % Structure length
B_struct      = Structure_data{2}(2); % Structure breadth
D_struct      = Structure_data{2}(3); % Structure diameter
Psi_struct    = Structure_data{3};    % Structure course
t_cr          = Structure_data{4};    % Critical distance to level ice
R_excl        = Structure_data{5};    % Ships exclusion zone radius
d_floe_target = Structure_data{6};    % Target floe size

% READING ICE DRIFT DATA
V_drift                    = Ice_drift_data{2}; % Drift velocity
alpha_drift                = Ice_drift_data{3}; % Drift direction
alpha_drift_dot_max_deg_hr = Ice_drift_data{4}; % Drift direction change
                                                % rate (! in deg/hr)

% READING SHIP PARAMETERS
B_ship         = Ship_data{3};      % Ship's breadth
P_D_max_MW     = Ship_data{14}(1); % Ship's max delivered power (! in MW)
k_channel      = Ship_data{18};     % Channel width to ship's beam ratio
prop_mode      = Ship_data{21}(1); % Propeller(s) rotating mode :
                        % 1: All propellers are rotating
                        % 2: 2 pods, only the pod outside the turn rotates
                        % 3: 2 pods, only the pod inside the turn rotates

advance_mode  = Ship_data{21}(1); % Type of advance along the trajectory :
                  % 1: Keeping same power as in linear motion on the whole
                  %    trajectory, V decreases in turns and increases in
                  %    floes.
                  % 2: Keeping same velocity as in linear motion on the
                  %    whole trajectory, P_D increases in turns (< P_D_max)
                  %    and decreases in floes.

% READING OTHER PARAMETERS
dt            = Other_data{3};  % Time step

if V_drift == 0,
    disp('ERROR : Zero drift velocity.')
    error('SIMULATION ABORTED');
end;


%% ICE MANAGEMENT CALCULATIONS
Psi_struct_rad  = mod(Psi_struct,360)*pi/180;
alpha_drift_rad = mod(alpha_drift,360)*pi/180;

alpha_drift_dot_max_deg_s = alpha_drift_dot_max_deg_hr/3600; %[deg/s]

% Icebreaker channel width
W_channel = B_ship*k_channel;

% 1st estimation of the turning radius
R_turn_estim = 0.5*(d_floe_target+W_channel);

% Angle related to the maximum admissible drift direction change
theta_sect = 90 - ...
             0.5*alpha_drift_dot_max_deg_s*(R_excl+R_turn_estim)/V_drift;
theta_sect_rad = theta_sect*pi/180;

% Width of area to be managed
disp('Calculation of managed area width')
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    W_area = D_struct + 2*t_cr + 2*(R_excl+R_turn_estim)...
             /tan(theta_sect_rad);%Width of managed area
    W_area_traj = W_area - W_channel; % Width of trajectory
elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    alpha_1_rad = alpha_drift_rad - Psi_struct_rad;
    alpha_2_rad = atan((B_struct/2+t_cr)/(L_struct/2+t_cr));
    if (mod(abs(alpha_1_rad*180/pi-270),360) < alpha_2_rad*180/pi) ||...
       (mod(abs(alpha_1_rad*180/pi-90),360) < alpha_2_rad*180/pi),
        OC = 0;
        theta_sect_rad_12 = (90-0.5*alpha_drift_dot_max_deg_s...
                            *(R_excl+B_struct/2 + t_cr)/V_drift)*pi/180;
        theta_sect_rad_1 = theta_sect_rad_12;
        theta_sect_rad_2 = theta_sect_rad_12;
        % Width of  the managed area
        W_area = abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
            + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
            + 2*t_cr...
            + 2*(R_excl + B_struct/2 + t_cr)/tan(theta_sect_rad_12);
        % Width of the trajectory
        W_area_traj = W_area - W_channel;

    elseif (mod(abs(alpha_1_rad*180/pi-0),360) < alpha_2_rad*180/pi) ||...
           (mod(abs(alpha_1_rad*180/pi-180),360) < alpha_2_rad*180/pi),
        OC = 0;
        theta_sect_rad_12 = (90-0.5*alpha_drift_dot_max_deg_s...
                            *(R_excl+L_struct/2+t_cr)/V_drift)*pi/180;
        theta_sect_rad_1 = theta_sect_rad_12;
        theta_sect_rad_2 = theta_sect_rad_12;
        % Width of  the managed area
        W_area = abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
            + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
            + 2*t_cr...
            + 2*(R_excl + L_struct/2 + t_cr)/tan(theta_sect_rad_12);
        % Width of the trajectory
        W_area_traj = W_area - W_channel;
    else
        OC = (sqrt((B_struct/2+t_cr)^2 ...
             + (L_struct/2+t_cr)^2))*cos(abs(alpha_1_rad)+alpha_2_rad);
        theta_sect_rad_1 = (90-0.5*alpha_drift_dot_max_deg_s...
                            *(R_excl+R_turn_estim-OC)/V_drift)*pi/180;
        theta_sect_rad_2 = (90-0.5*alpha_drift_dot_max_deg_s...
                            *(R_excl+R_turn_estim+OC)/V_drift)*pi/180;
        % Width of  the managed area
        W_area = abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
            + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
            + 2*t_cr...
            + (R_excl + R_turn_estim - OC)/tan(theta_sect_rad_1)...
            + (R_excl + R_turn_estim + OC)/tan(theta_sect_rad_2);
        % Width of the trajectory
        W_area_traj = W_area - W_channel;
    end;
end;


disp('Calculation of power level:')
disp('- Turning estimation')
turn_sign = +1;    % +1 for turn to port
turn_floes = 0;
V_turning_vect_init = [.1:.15:0.85, 1:.25:1.75, 2:.5:3];              %[m/s]
V_turning_vect      = horzcat(V_turning_vect_init, zeros(1,100));     %[m/s]
R_turning_vect      = zeros(1, length(V_turning_vect));
P_D_vect            = zeros(1, length(V_turning_vect));
beta_prop_deg_vect  = zeros(1, length(V_turning_vect));
beta_drift_deg_vect = zeros(1, length(V_turning_vect));

for i = 1:length(V_turning_vect_init),
    disp([' ',num2str(i), '/', num2str(length(V_turning_vect_init))])
    %Turning radius, for updrift turn
    R_turning_vect(i) = (0.5*(d_floe_target+W_channel))...
                        /(1+(pi/2)*V_drift/V_turning_vect(i));
                % NB : a downdrift turn would allow a larger R_turning, but
                % the ice management trajectory requires an updrift turn

    [P_D_vect(i), beta_prop_deg_vect(i), beta_drift_deg_vect(i)] =...
                        Calc_circle_V(R_turning_vect(i), V_turning_vect(i),...
                        turn_sign, prop_mode, turn_floes, fig_local);
end;
disp('  extension')
while P_D_vect(i) < 10*P_D_max_MW*1e6,
    i=i+1;
    V_turning_vect(i) = V_turning_vect(i-1) + 1;
    R_turning_vect(i) = (0.5*(d_floe_target+W_channel))...
                        /(1+(pi/2)*V_drift/V_turning_vect(i));
    [P_D_vect(i), beta_prop_deg_vect(i), beta_drift_deg_vect(i)] =...
                        Calc_circle_V(R_turning_vect(i), V_turning_vect(i),...
                        turn_sign, prop_mode, turn_floes, fig_local);
end;
i = i+1;
R_turning_vect(i:end)      = [];
V_turning_vect(i:end)      = [];
beta_prop_deg_vect(i:end)  = [];
beta_drift_deg_vect(i:end) = [];
P_D_vect(i:end)            = [];

i=1;
while i <= length(V_turning_vect), % Removing the NaN results
    if isnan(P_D_vect(i)),
        R_turning_vect(i)        = [];
        V_turning_vect(i)        = [];
        beta_prop_deg_vect(i)    = [];
        beta_drift_deg_vect(i)   = [];
        P_D_vect(i)              = [];
    else
        i = i+ 1;
    end;
end;


if fig_local,
    figure,
    subplot(1,3,1)
    plot(V_turning_vect, P_D_vect/1e6)
    xlabel('V turning [m/s]', 'Fontsize', 12)
    ylabel('P_D turning [MW]', 'Fontsize', 12)
    axis([0 V_turning_vect(end) 0 2*P_D_max_MW])

    subplot(1,3,2)
    plot(R_turning_vect, P_D_vect/1e6)
    xlabel('R turning [m]', 'Fontsize', 12)
    ylabel('P_D turning [MW]', 'Fontsize', 12)
    axis([0 R_turning_vect(end) 0 2*P_D_max_MW])

    subplot(1,3,3)
    plot(V_turning_vect, R_turning_vect)
    xlabel('V turning [m/s]', 'Fontsize', 12)
    ylabel('R turning [m]', 'Fontsize', 12)
    axis([0 V_turning_vect(end) 0 R_turning_vect(end)])
end;

%%
disp('- iterative calculation')

FCT_P_D = @(P_D) Calc_P_D_sector(P_D, W_channel, W_area_traj,...
                        R_turning_vect, V_turning_vect, P_D_vect,...
                        advance_mode);

% Lower range value
if advance_mode == 1, %Turning at same power as linear motion
    P_D_range_min = 1;
    fct_P_D_min = FCT_P_D(P_D_range_min);
    while isinf(fct_P_D_min)||isnan(fct_P_D_min)||~isreal(fct_P_D_min),
        P_D_range_min = 1.2*P_D_range_min;
        fct_P_D_min = FCT_P_D(P_D_range_min);
    end;
elseif advance_mode == 2,%Turning at same velocity as linear motion
    % Considering the lowest calculated turning velocity, we take the
    % linear motion power level for the same velocity.
    FCT_P_D_range_min = @(P_D_range_min) min(V_turning_vect)...
        - Equil_level_ice(P_D_range_min, 0, false, false);
    P_D_range_min = 1.0001*fzero(FCT_P_D_range_min,[0 10*P_D_max_MW*1e6]);
end;


% Upper range value
% We take the highest power value giving a realistic value
P_D_range_max = min(10*P_D_max_MW*1e6, P_D_vect(end));
fct_P_D_max = FCT_P_D(P_D_range_max);
while isinf(fct_P_D_max)||isnan(fct_P_D_max)||~isreal(fct_P_D_max)||...
      sign(fct_P_D_max) == sign(FCT_P_D(P_D_range_min)),
    P_D_range_max = 0.8*P_D_range_max;
```

```
    fct_P_D_max = FCT_P_D(P_D_range_max);
    if (P_D_range_max < 0.01*P_D_range_min) ||...
       (P_D_range_max < min(P_D_vect)),
        disp('ERROR : Unable to compute a power level.')
        error('SIMULATION ABORTED');
    end;
end;

%Resolution
P_D = fzero(FCT_P_D,[P_D_range_min P_D_range_max]);

disp_eq = false;
[V_lvl_ice, T_net_lvl, N_calc_lvl] = Equil_level_ice(P_D, 0,...
                                                     disp_eq, false);

[t_cycle_0, x_A, y_A, gamma, V_turning] = Calc_P_D_sector(P_D,...
                W_channel, W_area_traj, R_turning_vect, V_turning_vect,...
                P_D_vect, advance_mode);

R_turning      = interp1(V_turning_vect, R_turning_vect, V_turning);
P_D_turning    = interp1(V_turning_vect, P_D_vect, V_turning);
beta_prop_deg  = interp1(V_turning_vect, beta_prop_deg_vect, V_turning);
beta_drift_deg = interp1(V_turning_vect, beta_drift_deg_vect, V_turning);

% IF INSUFFICIENT POWER, COMPUTE MIN MANAGEABLE FLOE SIZE
disp('Results:')
if P_D > P_D_max_MW*1e6,
    disp(['Insufficient ship power. ',...
          'Calculation of minimum manageable floe size...'])
  P_D_max = P_D_max_MW*1e6;
  % Level ice velocity
  [V_lvl_ice_P_D_max, T_net_lvl_P_D_max, N_calc_lvl_P_D_max] = ...
           Equil_level_ice(P_D_max, 0, false, false);

    % Velocity for turning at maximum power level
    k=0;
    while P_D_vect(end-k) > (P_D_max_MW*1e6),
        k = k+1;
        if k == length(P_D_vect),
            disp(['ERROR : Unable to find a turning velocity for the ',...
                  'maximum power level. Unsufficient ice breaking ',...
                  'power.'])
            error('SIMULATION ABORTED');
        end;
    end;
    V_turning_P_D_max = interp1(P_D_vect((end-k):(end-k+1)),...
        V_turning_vect((end-k):(end-k+1)),...
        P_D_max_MW*1e6);
    if isnan(V_turning_P_D_max),
        disp(['ERROR : Unable to find a turning velocity for the',...
              ' maximum power level.'])
        error('SIMULATION ABORTED');
    end;

    % Width of area to be managed
    if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
        % Structure is circular
        W_area_cst = D_struct + 2*t_cr...
                   + 2*(R_excl+(D_struct+2*t_cr)/2)/tan(theta_sect_rad)...
                   - W_channel;
        d_floe_target_min = -W_channel...
                          + W_area_cst/(V_lvl_ice_P_D_max*(1/V_drift...
                          -(1/(V_lvl_ice_P_D_max*tan(theta_sect_rad))+...
                          pi/(2*V_turning_P_D_max))/(1+...
                          (pi/2)*V_drift/V_turning_P_D_max)));
    elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0);
        % Structure is rectangular
        if (mod(abs(alpha_1_rad*180/pi-270),360)<alpha_2_rad*180/pi)||...
           (mod(abs(alpha_1_rad*180/pi-90), 360)<alpha_2_rad*180/pi)||...
           (mod(abs(alpha_1_rad*180/pi-0),  360)<alpha_2_rad*180/pi)||...
           (mod(abs(alpha_1_rad*180/pi-180),360)<alpha_2_rad*180/pi),
            % In these cases, W_area is independent of R_turning
            d_floe_target_min = - W_channel...
                              + W_area_traj/(V_lvl_ice_P_D_max...
                              *(1/V_drift - pi/(2*V_turning_P_D_max...
                              *(1+(pi/2)*V_drift/V_turning_P_D_max))));
        else
            W_area_cst=...
                    abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
                  + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
                  + 2*t_cr...
                  + 2*R_excl/tan(0.5*(theta_sect_rad_1...
                  + theta_sect_rad_2));

            d_floe_target_min =-W_channel...
                              +W_area_cst/(V_lvl_ice_P_D_max*(1/V_drift...
                              -(1/(V_lvl_ice_P_D_max*tan(0.5 ...
                              *(theta_sect_rad_1+theta_sect_rad_2)))...
                              +pi/(2*V_turning_P_D_max))/(1+(pi/2)...
                              *V_drift/V_turning_P_D_max)));
        end;
    end;
end;

% COMMENTS ABOUT THE CALCULATION
if P_D <= P_D_max_MW*1e6,
  V_lvl_kts      = V_lvl_ice*1.94384449;
    V_turning_kts = V_turning*1.94384449;
    % Checking if close to maximum delivered power
    if P_D >= 0.9*P_D_max_MW*1e6,
        safety_new = 100*(P_D_max_MW*1e6 - P_D)/(P_D_max_MW*1e6);
        disp(['WARNING : Ice management is only possible with ',...
              num2str(safety_new, 3),...
              '% of security allowance on the delivered power.'])
    end;
    % Display calculation results
    disp('- Required delivered power : ')
    disp([' * Linear motion: ',num2str(P_D/1e6, 3), ' MW'])
    switch advance_mode,
        case 1, %Keeping same power as in linear motion, V decreases
            if P_D_turning == min(P_D_vect),
                disp([' * Turning:        ', num2str(P_D_turning/1e6,3),...
                      ' MW (min achievable turning power)'])
            else
                disp([' * Turning:        ', num2str(P_D_turning/1e6,3),...
                      ' MW (same as in linear motion)'])
            end;
        case 2, %Keeping same velocity as in linear motion, P_D increases
            if P_D_turning == P_D_max_MW*1e6,
                disp([' * Turning:        ', num2str(P_D_turning/1e6,3),...
                      ' MW (max delivered power)'])
            else
                disp([' * Turning:        ', num2str(P_D_turning/1e6,3),...
                      ' MW'])
            end;
    end;
    disp('- Level ice velocities :')
    disp([' * Linear motion : ', num2str(V_lvl_ice, 3), ' m/s (',...
          num2str(V_lvl_kts,3),' kts)'])
    switch advance_mode,
        case 1, %Keeping same power as in linear motion, V decreases
            if P_D_turning == min(P_D_vect),
                disp([' * Turning :        ', num2str(V_turning, 3),...
                      ' m/s (', num2str(V_turning_kts,3),...
                      ' kts) (at min achievable turning power)'])
            else
                disp([' * Turning :        ', num2str(V_turning, 3),...
                      ' m/s (', num2str(V_turning_kts,3),...
                      ' kts) (at same power as in linear motion)'])
            end;
        case 2, %Keeping same velocity as in linear motion, P_D increases
            if P_D_turning == P_D_max_MW*1e6,
                disp([' * Turning :        ', num2str(V_turning, 3),...
                      ' m/s (', num2str(V_turning_kts,3),...
                      ' kts) (at max delivered power)'])
            else
                disp([' * Turning :        ', num2str(V_turning, 3),...
                      ' m/s (', num2str(V_turning_kts,3),...
                      ' kts) (same as linear motion)'])
            end
    end;
    disp(['- Turning radius : ', num2str(R_turning,3), ' m'])
    switch prop_mode,
        case 1, % All propellers are rotating
            disp(['- Propeller(s) angle : +/- ',...
                  num2str(abs(beta_prop_deg),3), ' deg'])
        case 2, % 2 pods, only the pod outside the turn rotates
            disp(['- Propellers angles : ', num2str(beta_prop_deg,3),...
                  ' deg (outside prop.), ', num2str(0.0,3),...
                  ' deg (inside prop.)'])
        case 3, % 2 pods, only the pod inside the turn rotates
            disp(['- Propellers angles : ', num2str(0.0,3),...
                  ' deg (outside prop.), ', num2str(beta_prop_deg,3),...
                  ' deg (inside prop.)'])
    end;

else
  disp(['SIMULATION ABORTED : Insufficient Icebreaking power. ',...
        'Unable to manage the ice.'])
  disp( 'Possible solutions :')
      disp(['- increase icebreaking power up to at least ',...
            num2str(P_D/1e6, 3), ' MW, '])
  disp( '- use more than one management vessel,')
  disp(['- increase target floe size above ',...
        num2str(d_floe_target_min, 3),' m.'])
      error('SIMULATION ABORTED');
end;


%% TRAJECTORY CALCULATION
% Length of area to be managed
L_area = 2*R_turning;%d_floe_target+W_channel;

traj_init = [x_A; y_A];
dist_0 = R_excl+R_turning+W_channel/2;

% Rotation matrices
Rot_psi = [cos(Psi_struct_rad) -sin(Psi_struct_rad);...
    sin(Psi_struct_rad)  cos(Psi_struct_rad)];      %for Psi_struct
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];    %for alpha_drift

l_h      = ceil((2*sqrt(x_A^2+y_A^2))/V_lvl_ice/dt);
l_180g   = ceil((pi+2*gamma)*R_turning/V_turning/dt);

traj_0_t = zeros(2, 2*l_h + 2*l_180g);
traj_0_t(:,1) = traj_init;


n=1;

% perpendicular straight line, decreasing x
end_str_line_h1 = -x_A;

v_x = cos(atan(abs(y_A/x_A)));
v_y = sin(atan(abs(y_A/x_A)));

while traj_0_t(1,n) < end_str_line_h1,
      n =n+1;
  traj_0_t(:,n)=traj_0_t(:,n-1)+[v_x;v_y]*V_lvl_ice*dt;
end;
ex_h1 = traj_0_t(1,n)-end_str_line_h1; % exceedance


% turning (2*pi-2*gamma)deg
traj_0_t(:,n:(n+l_180g-1)) =  (traj_0_t(:,n)-[ex_h1;0])*ones(1,l_180g)...
        + [R_turning*...
        (cos((pi/2+gamma):(-(pi+2*gamma)/(l_180g-1)):(-pi/2-gamma))...
        -cos(pi/2+gamma));...
        R_turning*...
        (sin((pi/2+gamma):(-(pi+2*gamma)/(l_180g-1)):(-pi/2-gamma))...
        -sin(pi/2+gamma))];
n=n+l_180g-1;

% perpendicular straight line, increasing x
end_str_line_h2 = x_A;
while traj_0_t(1,n) > end_str_line_h2,
      n =n+1;
  traj_0_t(:,n)=traj_0_t(:,n-1)+[-v_x;v_y]*V_lvl_ice*dt;
end;
ex_h2 = traj_0_t(1,n)-end_str_line_h2; % exceedance

% turning (2*pi-2*gamma)deg
traj_0_t(:,n:(n+l_180g-1)) =  (traj_0_t(:,n)-[ex_h2;0])*ones(1,l_180g)...
        + [R_turning*...
        (cos((pi/2-gamma):((pi+2*gamma)/(l_180g-1)):(3*pi/2+gamma))...
        -cos(pi/2-gamma));...
        R_turning*...
        (sin((pi/2-gamma):((pi+2*gamma)/(l_180g-1)):(3*pi/2+gamma))...
        -sin(pi/2-gamma))];
n=n+l_180g-1;

traj_0_t(:,(n+1):end)=[];

% Offset due to asymmetry in the case of rectangular structure
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    e_h = 0;
elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    e_h = (sign(alpha_1_rad))*((R_excl + R_turning - OC)...
          /tan(theta_sect_rad_2)...
           - (R_excl + R_turning + OC)/tan(theta_sect_rad_1))/2;
end;

traj_0_t(1,:) = traj_0_t(1,:) + e_h;

% traj_t = zeros(size(traj_0_t));
% for k=1:length(traj_0_t(1,:)),
```

```
%     traj_t(:,k) = traj_0_t(:,k)'*Rot_alpha;
% end;
%
% traj_t(1,:) =   traj_t(1,:)...
%                 -(R_excl+R_turning+W_channel/2)*sin(alpha_drift_rad);
% traj_t(2,:) =   traj_t(2,:)...
%                 -(R_excl+R_turning+W_channel/2)*cos(alpha_drift_rad);


%% VIEW

% NB : the structure centre is in (0,0)

% Structure and critical area
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    draw_struct = [(D_struct/2)*cos(0:(pi/50):(2*pi));...
        (D_struct/2)*sin(0:(pi/50):(2*pi))];
    draw_crit   = [(D_struct/2+t_cr)*cos(0:(pi/50):(2*pi));...
        (D_struct/2+t_cr)*sin(0:(pi/50):(2*pi))];

elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    draw_struct_0 = [-B_struct/2, -B_struct/2, B_struct/2, B_struct/2,...
        -B_struct/2,;...
        -L_struct/2,  L_struct/2,  L_struct/2, -L_struct/2, -L_struct/2,];

    draw_crit_0   = [-B_struct/2+t_cr*sin(-pi:(pi/50):(-pi/2)),...
        -B_struct/2+t_cr*sin((-pi/2):(pi/50):0),...
        B_struct/2+t_cr*sin(0:(pi/50):(pi/2)),...
        B_struct/2+t_cr*sin((pi/2):(pi/50):pi),...
        -B_struct/2;...
        -L_struct/2+t_cr*cos(-pi:(pi/50):(-pi/2)),...
        L_struct/2+t_cr*cos((-pi/2):(pi/50):0),...
        L_struct/2+t_cr*cos(0:(pi/50):(pi/2)),...
        -L_struct/2+t_cr*cos((pi/2):(pi/50):pi),...
        -L_struct/2-t_cr];

    draw_struct = zeros(2,5);
    draw_crit   = zeros(2,length(draw_crit_0(1,:)));
    for k=1:5,
        draw_struct(:,k) = draw_struct_0(:,k)'*Rot_psi;
    end;
    for k=1:length(draw_crit_0(1,:)),
        draw_crit(:,k) = draw_crit_0(:,k)'*Rot_psi;
    end;
end;

% Exclusion zone
draw_excl_zone = [R_excl*cos(0:(pi/50):(2*pi));...
                  R_excl*sin(0:(pi/50):(2*pi))];

% Ice management area
draw_area_0 = [-W_area/2, W_area/2, W_area/2,-W_area/2,...
               -W_area/2;...
               -R_turning-W_channel/2, -R_turning-W_channel/2,...
                R_turning+W_channel/2, R_turning+W_channel/2,...
               -R_turning-W_channel/2];
draw_area_0(1,:) = draw_area_0(1,:) + e_h;

% Channel
draw_channel_0 =...
[draw_area_0(1,1),   draw_area_0(1,1),...
 draw_area_0(1,2),   draw_area_0(1,2);...
 draw_area_0(2,1)-4*R_excl, draw_area_0(2,1)+4*R_excl,...
 draw_area_0(2,2)+4*R_excl, draw_area_0(2,2)-4*R_excl];

draw_channel = zeros(2,length(draw_channel_0(1,:)));
draw_area    = zeros(2,length(draw_area_0(1,:)));

for k=1:length(draw_channel_0(1,:)),
    draw_channel(:,k) = draw_channel_0(:,k)'*Rot_alpha;
end;
for k=1:length(draw_area(1,:)),
    draw_area(:,k)    = draw_area_0(:,k)'*Rot_alpha;
end;



draw_area(1,:) =   draw_area(1,:) - dist_0*sin(alpha_drift_rad);
draw_area(2,:) =   draw_area(2,:) - dist_0*cos(alpha_drift_rad);


% Representation
n_traj      = 1;
frame_freq  = -1;%Value for representation only in Simulation_calc funct.
draw_labels = true;
mov         = {false};

X_lims_add = [];
Y_lims_add = [];

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone, draw_channel_0,...
                    draw_area, alpha_drift, dist_0);
end;

%% SIMULATION WITH CONSTANT DRIFT DIRECTION
n_traj      = 2;
frame_freq  = 25;
draw_labels = true;
if movie_rec,
    mov = {true; 'IM_linear'; 18};
else
    mov = {false};
end;

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
end;

%% FLOE SIZE ANALYSIS
t_cycle   = dt*length(traj_0_t);
fig_local = true;
fig_main  = true;


%Specific simulation for floe analysis (without display)
n_traj      = 20;
frame_freq  = 0;
draw_labels = false;
mov         = {false};
X_lims_add  = [];
Y_lims_add  = [];
[ice_ch_1, ice_ch_2] = Simulation_calc(traj_0_t, n_traj, frame_freq,...
                    draw_labels, mov, W_channel, L_area, W_area, X_lims_add,...
                    Y_lims_add, draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
% Floe analysis
Floe_size_analysis_sector(ice_ch_1, ice_ch_2, t_cycle, e_h, fig_local,...
                                                          fig_main);

%% SIMULATION WITH VARYING DRIFT DIRECTION

n_traj      = 20*.3;
frame_freq  = 80;
draw_labels = true;
if movie_rec,
    mov = {true; 'IM_circular_var'; 18};
else
    mov = {false};
end;

X_lims_add = [];
Y_lims_add = [];

if frame_freq == -1,
    k_end = 1;
else
    k_end = floor(n_traj*length(traj_0_t));
end

k_slope = k_end*0.6;

alpha_step = alpha_drift_dot_max_deg_s*dt;
alpha_max  = alpha_step*k_slope;

alpha_deg_t = alpha_drift + [zeros(1,k_end*0.2)...
                0:alpha_step:alpha_max alpha_max*ones(1,k_end*0.2+1)];

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                draw_struct, draw_crit, draw_excl_zone,...
                draw_channel_0, draw_area, alpha_deg_t, dist_0);
end;
```

### *D.7.4 Calc_P_D_sector.m*

```
function [t_cycle_0, x_A, y_A, gamma, V_turning] = Calc_P_D_sector(P_D,...
                  W_channel, W_area_traj, R_turning_vect, V_turning_vect,...
                  P_D_vect, turning_mode)
%
% This function is used in an iterative process calculating the required
% power level for sector ice management. The objective is to find the
% power level giving t_cycle_0 = 0 (see Ice_management_sector.m).
% The turning ability has been previously estimated in the vectors
% R_turning_vect, V_turning_vect and P_D_vect.
%
% INPUTS :
% P_D               [W]      Ship's delivered power
% W_channel         [m]      Width of ship's broken channel
% W_area_traj       [m]      Width of the sector ice management trajectory
% R_turning_vect    [m]      Turning radius vector for turning estimation
% V_turning_vect    [m/s]  Turning velocity vector for turning estimation
% P_D_vect          [W]      Delivered power vector for turning estimation
% turning_mode      [-]      Type of turning :
%                              1: Keeping same power as in linear motion, V
%                                 decreases
%                              2: Keeping same velocity as in linear motion,
%                                 P_D increases
%
% OUTPUTS:
% t_cycle_0         [s]      Target value (looking for t_cycle_0 to be zero)
% x_A               [m]      x-coord. of turning starting point for traj.
% y_A               [m]      y-coord. of turning starting point for traj.
% gamma             [deg]    Angle of the turning starting point
% V_turning         [m/s]   Turning velocity
%

global Ship_data
global Structure_data
global Ice_drift_data


% READING STRUCTURE PARAMETERS
d_floe_target = Structure_data{6}; % Target floe size

% READING ICE DRIFT DATA
V_drift       = Ice_drift_data{2}; % Drift velocity

% READING SHIP PARAMETERS
P_D_max_MW    = Ship_data{14}(1);% Ship's max delivered power (! in MW)


% Cycle time, delivered power level and level ice velocity
% LEVEL ICE VELOCITY
V_lvl_ice = Equil_level_ice(P_D, 0, false, false);

% REQUIRED TURNING RADIUS AND DELIVERED POWER

% Calculating velocity for turning at maximum power level
k=0;
while P_D_vect(end-k) > (P_D_max_MW*1e6),
    k = k+1;
    if k == length(P_D_vect),
        disp(['ERROR : Unable to find a turning velocity for the ',...
              'maximum power level. Please increase ice breaking ',...
              'power up to at least ',num2str(min(P_D_vect/1e6),3),' MW.'])
        error('SIMULATION ABORTED');
    end;
end;
V_turning_P_D_max = interp1(P_D_vect((end-k):(end-k+1)),...
                            V_turning_vect((end-k):(end-k+1)),...
                            P_D_max_MW*1e6);
if isnan(V_turning_P_D_max),
    disp(['ERROR : Unable to find a turning velocity for the maximum ',...
          'power level.'])
    error('SIMULATION ABORTED');
end;



if turning_mode == 1, %TURNING AT SAME POWER AS IN LINEAR MOTION
                      %(or, if impossible, at the lowest turning power)
    P_D_turning    = P_D;
    V_turning_calc = zeros(1, length(P_D_vect)-1);

    % In this mode, one power level may be associated to 1, 2 or more
    % turning configurations. We look for all these configurations.
    for i = 1:(length(P_D_vect)-1),
        V_turning_calc(i) = interp1(P_D_vect(i:(i+1)),...
                            V_turning_vect(i:(i+1)), P_D_turning);
    end;
    i = 1;
    while i <= length(V_turning_calc),
        if isnan(V_turning_calc(i)),
            V_turning_calc(i) = [];
            i = i-1;
        end;
        i = i+1;
    end;
    nbr_config = length(V_turning_calc); % Number of configurations
    R_turning  = interp1(V_turning_vect, R_turning_vect, V_turning_calc);

    if nbr_config == 0,
        if P_D_turning < min(P_D_vect),
            % Turning at the lowest achievable turning power
            [P_D_turning, k_min_P_D]  = min(P_D_vect);
            V_turning_calc = V_turning_vect(k_min_P_D);
            R_turning      = interp1(V_turning_vect, R_turning_vect,...
                                     V_turning_calc);
            nbr_config     = 1;
        elseif P_D_turning < max(P_D_vect),
            disp(['ERROR : Unable to find a turning velocity for the',...
                ' requested power level. Excessive power level.'])
            error('SIMULATION ABORTED');
        else
            disp(['ERROR : Unable to find a turning velocity for the',...
                ' requested power level.'])
            error('SIMULATION ABORTED');
        end;
    end;

    W_area_st       = zeros(1,nbr_config);
    x_A_calc        = zeros(1,nbr_config);
    y_A_calc        = zeros(1,nbr_config);
    gamma_calc      = zeros(1,nbr_config);
    t_cycle_0_calc  = zeros(1,nbr_config);
    for n=1:nbr_config,
        % Width of the linear motion part of the area to be managed
        W_area_st(n) = W_area_traj - 2*R_turning(n);
        x_A0 = (R_turning(n)^2)*2/W_area_st(n);
        y_A0 = (-R_turning(n)*2/W_area_st(n))...
                *sqrt((W_area_st(n)/2)^2-R_turning(n)^2);
        x_A_calc(n)    = x_A0 - W_area_st(n)/2;
        y_A_calc(n)    = y_A0;
        gamma_calc(n) = atan(abs(x_A0/y_A0));

        t_cycle_traj = 4*sqrt(x_A_calc(n)^2+y_A_calc(n)^2)/V_lvl_ice...
            + (2*pi+4*gamma_calc(n))*R_turning(n)/V_turning_calc(n);

        % Looking for the following variable to be 0 :
        t_cycle_0_calc(n) = t_cycle_traj...
                            - 2*(d_floe_target+W_channel)/V_drift;
        if ~isreal(t_cycle_0_calc(n)),
            t_cycle_0_calc(n) = NaN;
        end;
    end;
    [t_cycle_0_min_abs, n_min] = min(abs(t_cycle_0_calc));
    t_cycle_0 = t_cycle_0_calc(n_min);
    V_turning = V_turning_calc(n_min);
    x_A       = x_A_calc(n_min);
    y_A       = y_A_calc(n_min);
    gamma     = gamma_calc(n_min);


elseif turning_mode == 2,%TURNING AT SAME VELOCITY AS IN LINEAR MOTION
    V_turning   = V_lvl_ice;
    R_turning   = interp1(V_turning_vect, R_turning_vect, V_turning);
    P_D_turning = interp1(V_turning_vect, P_D_vect, V_turning);

    if V_turning >= V_turning_P_D_max,
        %The required turning velocity is too large
        V_turning   = V_turning_P_D_max; %turning at maximum power level
%         P_D_turning = P_D_max_MW*1e6;
    elseif isnan(P_D_turning),
        %The required turning velocity is
        %-either zero (unsufficient power in level ice)
        %-either too low for turning at the given R_turning
        disp(['ERROR : Unable to find a delivered power for the',...
              ' requested turning velocity.'])
        error('SIMULATION ABORTED');
    end;

    % Width of the straight linear motion part of the area to be managed
    W_area_st = W_area_traj - 2*R_turning;

    x_A0 = (R_turning^2)*2/W_area_st;
    y_A0 = (-R_turning*2/W_area_st)*sqrt((W_area_st/2)^2-R_turning^2);
    x_A  = x_A0 - W_area_st/2;
    y_A  = y_A0;
    gamma= atan(abs(x_A0/y_A0));

    t_cycle_traj = 4*sqrt(x_A^2+y_A^2)/V_lvl_ice...
                   + (2*pi+4*gamma)*R_turning/V_turning;

    % Looking for the following variable to be 0 :
    t_cycle_0 = t_cycle_traj - 2*(d_floe_target+W_channel)/V_drift;

else
    disp('Unknown turning mode.')
    error('SIMULATION ABORTED')
end;

end
```

### *D.7.5 Ice_management_circular.m*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                          SIMULATION OF THE                                    %
%                     CIRCULAR ICE MANAGEMENT TECHNIQUE                         %
%                                                                               %
%                                                                               %
%                            Quentin HISETTE                                    %
%                                                                               %
%                                                                               %
%                             2013-2014                                         %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear all
close all
clc

fig_local = false;  % [Bool]  Activate/Deactivate local figures plotting
fig_main  = true;   % [Bool]  Activate/Deactivate main figures plotting
movie_rec = false;  % [Bool]  Activate/Deactivate movie recording

% READING INPUT FILES
disp('Reading input files')
Input_reading;
global Ship_data
global Structure_data
global Ice_drift_data
global Other_data

% READING STRUCTURE PARAMETERS
Ref           = Structure_data{1};     % Reference
L_struct      = Structure_data{2}(1); % Structure length
B_struct      = Structure_data{2}(2); % Structure breadth
D_struct      = Structure_data{2}(3); % Structure diameter
Psi_struct    = Structure_data{3};     % Structure course
t_cr      = Structure_data{4};     % Critical distance to level ice
R_excl        = Structure_data{5};     % Ships exclusion zone radius
d_floe_target = Structure_data{6};     % Target floe size

% READING ICE DRIFT DATA
V_drift                   = Ice_drift_data{2}; % Drift velocity
alpha_drift               = Ice_drift_data{3}; % Drift direction
alpha_drift_dot_max_deg_hr = Ice_drift_data{4}; % Drift direction change
                                                % rate (! in deg/hr)

% READING SHIP PARAMETERS
B_ship         = Ship_data{3};      % Ship's breadth
P_D_max_MW     = Ship_data{14}(1); % Ship's max delivered power (! in MW)
k_channel   = Ship_data{18};     % Channel width to ship's beam ratio
prop_mode      = Ship_data{21}(1); % Propeller(s) rotating mode :
                        % 1: All propellers are rotating
                        % 2: 2 pods, only the pod outside the turn rotates
                        % 3: 2 pods, only the pod inside the turn rotates

% READING OTHER PARAMETERS
dt             = Other_data{3};  % Time step

if V_drift == 0,
    disp('ERROR : Zero drift velocity.')
    error('SIMULATION ABORTED');
end;

%% ICE MANAGEMENT CALCULATIONS
disp('Calculation of trajectory radius')

Psi_struct_rad  = mod(Psi_struct,360)*pi/180;
alpha_drift_rad = mod(alpha_drift,360)*pi/180;

alpha_drift_dot_max_deg_s = alpha_drift_dot_max_deg_hr/3600; %[deg/s]


% Icebreaker channel width
W_channel = B_ship*k_channel;

% Width of area to be managed
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    FCT_R_traj = @(R_traj) - 2*R_traj  + D_struct   + 2*t_cr...
                           + 2*(R_excl + D_struct/2 +   t_cr...
                           + R_traj)/tan(pi/2  -...
                           alpha_drift_dot_max_deg_s*(pi/180)...
                           *(R_excl + D_struct + t_cr + R_traj)/V_drift);
    e_h = 0;

elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    alpha_1_rad = alpha_drift_rad - Psi_struct_rad;
    alpha_2_rad = atan((B_struct/2+t_cr)/(L_struct/2+t_cr));
    if (mod(abs(alpha_1_rad*180/pi-270),360) < alpha_2_rad*180/pi) ||...
       (mod(abs(alpha_1_rad*180/pi-90),360) < alpha_2_rad*180/pi),
        FCT_R_traj = @(R_traj) - 2*R_traj...
                     + abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
                     + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
                     + 2*t_cr...
                     + 2*(R_excl+B_struct/2+t_cr+R_traj)...
                     /tan((90-0.5*alpha_drift_dot_max_deg_s...
                     *(R_excl+B_struct/2+t_cr+R_traj)/V_drift)*pi/180);
        e_h = 0;

    elseif (mod(abs(alpha_1_rad*180/pi-0),360) < alpha_2_rad*180/pi) ||...
           (mod(abs(alpha_1_rad*180/pi-180),360) < alpha_2_rad*180/pi),
        FCT_R_traj = @(R_traj) - 2*R_traj...
                     + abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
                     + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
                     + 2*t_cr...
                     + 2*(R_excl+L_struct/2+t_cr+R_traj)...
                     /tan((90-0.5*alpha_drift_dot_max_deg_s...
                     *(R_excl+L_struct/2+t_cr+R_traj)/V_drift)*pi/180);
        e_h = 0;

    else
        OC = (sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
                *cos(abs(alpha_1_rad)+alpha_2_rad);
        FCT_R_traj = @(R_traj) - 2*R_traj...
                     + abs(L_struct*sin(alpha_drift_rad-Psi_struct_rad))...
                     + abs(B_struct*cos(alpha_drift_rad-Psi_struct_rad))...
                     + 2*t_cr...
                     + (R_excl...
                       -(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
                       *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
                       /tan((90-0.5*alpha_drift_dot_max_deg_s...
                       *(R_excl...
                       -(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
                       *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
                       /V_drift)*pi/180)...
                     + (R_excl...
                       +(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
                       *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
                       /tan((90-0.5*alpha_drift_dot_max_deg_s...
                       *(R_excl...
                       +(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
                       *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
                       /V_drift)*pi/180);

    end;
end;

%Interval for fzero(FCT_R_traj)
R_traj_min = B_ship/2;
R_traj_max = R_traj_min;
while sign(FCT_R_traj(R_traj_max)) == sign(FCT_R_traj(R_traj_min)),
    R_traj_max = 1.2*R_traj_max;
end;
% Calculation of R_traj
R_traj = fzero(FCT_R_traj, [R_traj_min R_traj_max]);

W_area_m    = 2*R_traj + W_channel; % Width of managed area

% Ship's velocity in the turn
V_traj = 0.001;
fl_1 = d_floe_target+1;
fl_2 = d_floe_target+1;
fl_3 = d_floe_target+1;
while (fl_1 > d_floe_target) || (fl_2 > d_floe_target) ||...
      (fl_3 > d_floe_target),
    V_traj = V_traj + 0.001;
    OF = V_drift*2*pi*R_traj/V_traj;
    OP = 2*R_traj - V_drift*pi*R_traj/V_traj;
    fl_1 = mod(OP, OF) - W_channel;
    fl_2 = OF - mod(OP, OF) - W_channel;
    fl_3 = (OF - W_channel)/sqrt(2);
end;

if fig_local,
    V_traj_vect = 0.1:0.01:6;
    fl_1_vect = zeros(1,length(V_traj_vect));
    fl_2_vect = zeros(1,length(V_traj_vect));
    fl_3_vect = zeros(1,length(V_traj_vect));
    for i = 1:length(V_traj_vect),
        OF = V_drift*2*pi*R_traj/V_traj_vect(i);
        OP = 2*R_traj - V_drift*pi*R_traj/V_traj_vect(i);
        fl_1_vect(i) = mod(OP, OF) - W_channel;
        fl_2_vect(i) = OF - mod(OP, OF) - W_channel;
        fl_3_vect(i) = (OF - W_channel)/sqrt(2);
    end;

    figure,
    plot(V_traj_vect, fl_1_vect, 'b')
    hold on,
    plot(V_traj_vect, fl_2_vect, 'r')
    hold on,
    plot(V_traj_vect, fl_3_vect, 'g')
    hold on,
    plot([V_traj_vect(1) V_traj_vect(end)], d_floe_target*[1 1], 'k')
    title('Looking for optimal velocity')
    set(gca, 'Fontsize', 12)
    legend('floe 1', 'floe 2', 'floe 3', 'target size')
    xlabel('$V_{traj}\,$ $[m/s]$', 'Fontsize', 12, ...
           'interpreter', 'latex')
    ylabel('\textit{Floe size} $[m]$', 'Fontsize', 12, ...
           'interpreter', 'latex')
end;

%% REQUIRED DELIVERED POWER
disp('Calculation of required power level')

turn_sign    = +1; % +1 for turn to port

% Turning configuration : P_D, propeller(s) angle and drift angle
turn_floes = false;
[P_D, beta_prop_deg, beta_drift_deg] = Calc_circle_V(R_traj, V_traj,...
                        turn_sign, prop_mode, turn_floes, fig_local);

%%
% IF INSUFFICIENT POWER, COMPUTE MIN MANAGEABLE FLOE SIZE AND MAXIMUM
% MANAGEABLE AREA
% - MINIMUM MANAGEABLE FLOE SIZE
if P_D > P_D_max_MW*1e6,
    disp('Insufficient ship power.')
    disp('* Calculation of minimum manageable floe size...')
    V_traj_P_D_max = Calc_circle_P_D(R_traj, P_D_max_MW*1e6,...
                     turn_sign, prop_mode, turn_floes, fig_local);
    if isnan(V_traj_P_D_max),
        disp(['ERROR : Unable to find a turning velocity for the',...
              ' maximum delivered power'])
        error('SIMULATION ABORTED');
    end;
    % Minimum manageable floe size
    d_floe_min = inf;
    for V_traj_min = V_traj_P_D_max:(-0.001):0.001,
        OF = V_drift*2*pi*R_traj/V_traj_min;
        OP = 2*R_traj - V_drift*pi*R_traj/V_traj_min;
        fl_1 = mod(OP, OF) - W_channel;
        fl_2 = OF - mod(OP, OF) - W_channel;
        fl_3 = (OF - W_channel)/sqrt(2);
        fl_max_k = max([fl_1, fl_2, fl_3]);
        if d_floe_min > fl_max_k,
            d_floe_min = fl_max_k;
            V_floe_min = V_traj_min;
        end;
    end;
end;

% - MAXIMUM MANAGEABLE AREA WIDTH
if P_D > P_D_max_MW*1e6,
    disp('* Calculation of maximum manageable area width...')
    calc      = true;
    R_traj_k = R_traj;
    P_D_k     = P_D;
    err       = inf;

    R_traj_vect = (0.1:.01:1)*R_traj;
    P_D_vect    = zeros(1, length(R_traj_vect));
    for k = 1:length(R_traj_vect),
        disp([num2str(k), '/', num2str(length(R_traj_vect))])
        % Ship's velocity in the turn
        V_traj_k = 0.001;
        fl_1 = d_floe_target+1;
        fl_2 = d_floe_target+1;
        fl_3 = d_floe_target+1;
        while (fl_1 > d_floe_target) || (fl_2 > d_floe_target) ||...
              (fl_3 > d_floe_target),
            V_traj_k = V_traj_k + 0.001;
            OF = V_drift*2*pi*R_traj_vect(k)/V_traj_k;
            OP = 2*R_traj_vect(k) - V_drift*pi*R_traj_vect(k)/V_traj_k;
```

```
            fl_1 = mod(OP, OF) - W_channel;
            fl_2 = OF - mod(OP, OF) - W_channel;
            fl_3 = (OF - W_channel)/sqrt(2);
        end;

        % Required delivered power
        turn_floes = false;
        P_D_vect(k) = Calc_circle_V(R_traj_vect(k), V_traj_k,...
                                turn_sign, prop_mode, turn_floes, fig_local);
    end;

    if fig_local,
        figure,
        plot(R_traj_vect, P_D_vect/1e6, 'b')
        hold on,
        plot([min(R_traj_vect) max(R_traj_vect)], [1 1]*P_D_max_MW, 'k')
        xlim([min(R_traj_vect) max(R_traj_vect)])
        set(gca, 'Fontsize', 12)
        legend('Required power level ', 'Ship max power level ')
        xlabel('$R_{traj}$ $[m]$', 'Fontsize', 12, 'Interpreter', 'latex')
        ylabel('$P_{D}$ $[MW]$', 'Fontsize', 12, 'Interpreter', 'latex')
        title('\textit{Calculation of maximum manageable area width}',...
              'Fontsize', 12, 'Interpreter', 'latex')

    end;

    err_P_D          = (P_D_vect - P_D_max_MW*1e6)/(P_D_max_MW*1e6);
    R_traj_max_vect = [];
    j = 1;
    while j <= length(err_P_D),
        if isnan(err_P_D(j)),
            err_P_D(j)     = [];
            R_traj_vect(j) = [];
            P_D_vect(j)    = [];
            j = j-1;
        elseif (j >=2) && sign(err_P_D(j)) ~= sign(err_P_D(j-1)),
            R_traj_max_vect = horzcat(R_traj_max_vect,...
                interp1(err_P_D((j-1):j), R_traj_vect((j-1):j),0));
        end;
        j = j+1;
    end;

    R_traj_max = max(R_traj_max_vect);
end;


%% COMMENTS ABOUT THE CALCULATION
if P_D <= P_D_max_MW*1e6,
  V_traj_kts = V_traj*1.94384449;
    % Checking if close to maximum delivered power
    if P_D >= 0.9*P_D_max_MW*1e6,
    safety_new = 100*(P_D_max_MW*1e6 - P_D)/(P_D_max_MW*1e6);
  disp(['WARNING : Ice management is only possible with ',...
        num2str(safety_new, 3),...
        '% of security allowance on the delivered power.'])
    end;
    % Display calculation results
  disp(['The required delivered power is : ',num2str(P_D/1e6,3), ' MW'])
  disp(['- Level ice velocity : V_lvl = ',...
        num2str(V_traj, 3), ' m/s (', num2str(V_traj_kts,3),...
        ' kts)'])
    disp(['- Turning radius : R_turning = ', num2str(R_traj, 3), ' m'])
    switch prop_mode,
        case 1, % All propellers are rotating
            disp(['- Propeller(s) angle = ', num2str(beta_prop_deg,3),...
                  ' deg'])
        case 2, % 2 pods, only the pod outside the turn rotates
            disp(['- Propellers angles = ', num2str(beta_prop_deg,3),...
                  ' deg (outside prop.), ', num2str(0.0,3),...
                  ' deg (inside prop.)'])
        case 3, % 2 pods, only the pod inside the turn rotates
            disp(['- Propellers angles = ', num2str(0.0,3),...
                  ' deg (outside prop.), ', num2str(beta_prop_deg,3),...
                  ' deg (inside prop.)'])
    end;
    disp(['- Drift angle = ', num2str(beta_drift_deg,3), ' deg'])
else
  disp(['SIMULATION ABORTED : Insufficient Icebreaking power. ',...
        'Unable to manage the ice.'])
    disp(['With the given vessel and the requested target floe size, ',...
        'it is only possible'])
    disp(['to manage ', num2str(R_traj_max+W_channel, 3),...
        'm of the requested width of ', num2str(R_traj+W_channel, 3),...
        'm. (',...
        num2str(100*(R_traj_max+W_channel)/(R_traj+W_channel), 3),...
        '%).'])
  disp( 'Possible solutions :')
    disp(['- increase icebreaking power up to at least ',...
        num2str(P_D/1e6, 3), ' MW, '])
  disp( '- use more than one management vessel,')
  disp(['- increase target floe size above ',...
         num2str(d_floe_min, 3),' m.'])
     error('SIMULATION ABORTED');
end;



%% TRAJECTORY CALCULATION
% Calculation of e_h
if ((D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0)) &&...
        ~((mod(abs(alpha_1_rad*180/pi-270),360)<alpha_2_rad*180/pi) ||...
          (mod(abs(alpha_1_rad*180/pi-90),360) <alpha_2_rad*180/pi))&&...
        ~((mod(abs(alpha_1_rad*180/pi-0),360)  <alpha_2_rad*180/pi) ||...
          (mod(abs(alpha_1_rad*180/pi-180),360)<alpha_2_rad*180/pi)),
    e_h = (sign(alpha_1_rad))*0.5*((R_excl...
        -(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
        *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
        /tan((90-0.5*alpha_drift_dot_max_deg_s...
        *(R_excl...
        -(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
        *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
        /V_drift)*pi/180)...
        -...
        (R_excl...
        +(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
        *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
        /tan((90-0.5*alpha_drift_dot_max_deg_s...
        *(R_excl...
        +(sqrt((B_struct/2+t_cr)^2 + (L_struct/2+t_cr)^2))...
        *cos(abs(alpha_1_rad)+alpha_2_rad)+R_traj)...
        /V_drift)*pi/180));
end;

traj_init   = [e_h; 0];
dist_0      = R_excl+2*R_traj+ W_channel/2;%Distance of the trajectory
                                          %centre to the structure centre
% Rotation matrices
Rot_psi = [cos(Psi_struct_rad) -sin(Psi_struct_rad);...
    sin(Psi_struct_rad)  cos(Psi_struct_rad)];          %for Psi_struct
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];        %for alpha_drift


l_360   = ceil(2*pi*R_traj/V_traj/dt);

traj_0_t = zeros(2, l_360);
traj_0_t(:,1) = traj_init;


n=1;

% turning 360 deg
traj_0_t(:,n:(n+l_360-1)) =  traj_0_t(:,n)*ones(1,l_360)...
                + [R_traj*(cos((-pi/2):(2*pi/(l_360-1)):(3*pi/2))-0);...
                   R_traj*(sin((-pi/2):(2*pi/(l_360-1)):(3*pi/2))+1)];
n=n+l_360-1;

traj_0_t(:,n:end)=[];


%% VIEW

% NB : the structure centre is in (0,0)

% Structure and critical area
if (D_struct ~= 0) && (L_struct == 0) && (B_struct == 0),
    % Structure is circular
    draw_struct = [(D_struct/2)*cos(0:(pi/50):(2*pi));...
        (D_struct/2)*sin(0:(pi/50):(2*pi))];
    draw_crit   = [(D_struct/2+t_cr)*cos(0:(pi/50):(2*pi));...
        (D_struct/2+t_cr)*sin(0:(pi/50):(2*pi))];

elseif (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
    % Structure is rectangular
    draw_struct_0 = [-B_struct/2, -B_struct/2,  B_struct/2, B_struct/2,...
        -B_struct/2,;...
        -L_struct/2,  L_struct/2,  L_struct/2, -L_struct/2, -L_struct/2,];

    draw_crit_0   = [-B_struct/2+t_cr*sin(-pi:(pi/50):(-pi/2)),...
        -B_struct/2+t_cr*sin((-pi/2):(pi/50):0),...
        B_struct/2+t_cr*sin(0:(pi/50):(pi/2)),...
        B_struct/2+t_cr*sin((pi/2):(pi/50):pi),...
        -B_struct/2;...
        -L_struct/2+t_cr*cos(-pi:(pi/50):(-pi/2)),...
        L_struct/2+t_cr*cos((-pi/2):(pi/50):0),...
        L_struct/2+t_cr*cos(0:(pi/50):(pi/2)),...
        -L_struct/2+t_cr*cos((pi/2):(pi/50):pi),...
        -L_struct/2-t_cr];

    draw_struct = zeros(2,5);
    draw_crit   = zeros(2,length(draw_crit_0(1,:)));
    for k=1:5,
        draw_struct(:,k) = draw_struct_0(:,k)'*Rot_psi;
    end;
    for k=1:length(draw_crit_0(1,:)),
        draw_crit(:,k) = draw_crit_0(:,k)'*Rot_psi;
    end;
end;

% Exclusion zone
draw_excl_zone = [R_excl*cos(0:(pi/50):(2*pi));...
                  R_excl*sin(0:(pi/50):(2*pi))];

% Ice management area
draw_area_0 = [-W_area_m/2, W_area_m/2,W_area_m/2,-W_area_m/2,...
               -W_area_m/2;...
               -W_area_m/2,-W_area_m/2,W_area_m/2, W_area_m/2,...
               -W_area_m/2];
draw_area_0(1,:) = draw_area_0(1,:) + e_h;

% Channel
draw_channel_0 =...
[draw_area_0(1,1),    draw_area_0(1,1),...
 draw_area_0(1,2),    draw_area_0(1,2);...
 draw_area_0(2,1)-4*(R_excl+R_traj),draw_area_0(2,1)+4*(R_excl+R_traj),...
 draw_area_0(2,2)+4*(R_excl+R_traj),draw_area_0(2,2)-4*(R_excl+R_traj)];

draw_channel = zeros(2,length(draw_channel_0(1,:)));
draw_area    = zeros(2,length(draw_area_0(1,:)));

for k=1:length(draw_channel_0(1,:)),
    draw_channel(:,k) = draw_channel_0(:,k)'*Rot_alpha;
end;
for k=1:length(draw_area(1,:)),
    draw_area(:,k)    = draw_area_0(:,k)'*Rot_alpha;
end;

draw_area(1,:) =   draw_area(1,:)...
                 -(R_excl+R_traj+ W_channel/2)*sin(alpha_drift_rad);
draw_area(2,:) =   draw_area(2,:)...
                 -(R_excl+R_traj+ W_channel/2)*cos(alpha_drift_rad);


% Representation
n_traj      = 1;
frame_freq  = -1;%Value for representation only in Simulation_calc funct.
draw_labels = ~false;
mov         = {false};

L_area = 2*R_traj;
W_area = 2*R_traj;

X_lims_add = [];
Y_lims_add = [];



if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
end;

%% SIMULATION WITH CONSTANT DRIFT DIRECTION
n_traj      = 5;
frame_freq  = 20;
draw_labels = true;
if movie_rec,
    mov = {true; 'IM_circular'; 18};
else
    mov = {false};
end;

L_area = 2*R_traj;
W_area = 2*R_traj;

X_lims_add = [];
Y_lims_add = [];
```

```
if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_drift, dist_0);
end;


%% FLOE SIZE ANALYSIS
t_cycle   = dt*length(traj_0_t);
fig_local = false;
fig_main  = true;

%Specific simulation for floe analysis (without display)
n_traj      = 20;
frame_freq  = 0;
draw_labels = false;
mov         = {false};

[ice_ch_1, ice_ch_2] = Simulation_calc(traj_0_t, n_traj, frame_freq,...
                draw_labels, mov, W_channel, L_area, W_area, X_lims_add,...
                Y_lims_add, draw_struct, draw_crit, draw_excl_zone,...
                draw_channel_0, draw_area, alpha_drift, dist_0);

% Floe analysis
Floe_size_analysis_circular(ice_ch_1, ice_ch_2, t_cycle, e_h, R_traj,...
                                                    fig_local, fig_main);
%% SIMULATION WITH VARYING DRIFT DIRECTION

n_traj      = 14.;
frame_freq  = 40;
draw_labels = true;
if movie_rec,
    mov = {true; 'IM_circular_var'; 18};
else
    mov = {false};
end;

L_area = 2*R_traj;
W_area = 2*R_traj;

X_lims_add = [];
Y_lims_add = [];


if frame_freq == -1,
    k_end = 1;
else
    k_end = floor(n_traj*length(traj_0_t));
end

k_slope = k_end*0.4;

alpha_step = alpha_drift_dot_max_deg_s*dt;
alpha_max  = alpha_step*k_slope;

alpha_deg_t = alpha_drift + [zeros(1,k_end*0.3)...
              0:alpha_step:alpha_max alpha_max*ones(1,k_end*0.3+1)];

if fig_main,
    Simulation_calc(traj_0_t, n_traj, frame_freq, draw_labels, mov,...
                    W_channel, L_area, W_area, X_lims_add, Y_lims_add,...
                    draw_struct, draw_crit, draw_excl_zone,...
                    draw_channel_0, draw_area, alpha_deg_t, dist_0);
end;
```

### *D.7.6 Simulation_calc.m*

```
function [ice_ch_1, ice_ch_2] = Simulation_calc(traj_0_t, n_traj,...
                    frame_freq, draw_labels, mov, W_channel, L_area,...
                    W_area, X_lims_add, Y_lims_add, draw_struct,...
                    draw_crit, draw_excl_zone, draw_channel_0,...
                    draw_area, alpha_deg_t, dist_0)
%
% Simulation of the ice management operation, with possible varying drift
% direction.
%
% INPUTS :
% traj_t          [m]   Coord. (x,y) of ship trajectory at each time step
% n_traj          [-]   Number of repetitions of the trajectory
% frame_freq      [-]   Frame frequency of the visualization
%                         "frame_freq =  0" to disable visualization
%                         "frame_freq = -1" to view config. w/o simulation
% draw_labels     [Bool] Activate/Deactivate labels on the visualization
% mov             [Bool] Activate/Deactivate movie recording :
%                         - Activate   : mov = {true; 'filename'; FPS}
%                         - Deactivate : mov = {false}
% W_channel       [m]   Width of ship's broken channel
% L_area          [m]   Length of managed area
% W_area          [m]   Width of managed area
% X_lims_add      [m]   Vector of additional X-axes limits (case-related)
% Y_lims_add      [m]   Vector of additional Y-axes limits (case-related)
% draw_struct     [m]     Structure representation
% draw_crit       [m]     Critical zone representation
% draw_excl_zone  [m]     Exclusion zone representation
% draw_channel    [m]     Managed channel representation
% draw_area       [m]     Managed area representation
% alpha_deg_t     [deg]   Time series of the evolution of the drift angle
%                         If single value, constant drift direction
% dist_0          [m]     Distance of trajectory centre to structure centre
%
% OUTPUTS :
% ice_ch_1        [m]     Coord. of one side of the broken channel
% ice_ch_2        [m]     Coord. of the other side of the broken channel
%

global Structure_data
global Ice_drift_data
global Other_data

% READING STRUCTURE PARAMETERS
Ref             = Structure_data{1};    % Reference
L_struct        = Structure_data{2}(1); % Structure length
B_struct        = Structure_data{2}(2); % Structure breadth
D_struct        = Structure_data{2}(3); % Structure diameter
Psi_struct      = Structure_data{3};    % Structure course
R_excl          = Structure_data{5};    % Ships exclusion zone radius

% READING ICE DRIFT DATA
V_drift         = Ice_drift_data{2}; % Drift velocity
alpha_drift     = Ice_drift_data{3}; % Drift direction

% READING OTHER PARAMETERS
dt              = Other_data{3};  % Time step

% ANGLES
Psi_struct_rad     = mod(Psi_struct,360)*pi/180;
alpha_drift_rad    = mod(alpha_drift,360)*pi/180;
alpha_drift_rad_t = alpha_deg_t*pi/180;


% CHECKING IF MOVIE CAN BE RECORDED
if mov{1} && frame_freq == -1,
    disp(['ERROR : Movie cannot be recorded without vizualisation.',...
          'Please use frame_freq >= 1.']);
    error('SIMULATION ABORTED')
end;

% AXES LIMITS
if frame_freq ~= 0,
    X_lims = sort([.7*R_excl*sin(alpha_drift_rad);...
                   -1.3*(sqrt((R_excl+L_area)^2+(W_area/2)^2))...
                                                   *sin(alpha_drift_rad);...
                   -0.8*R_excl;...
                    0.8*R_excl;...
                    1.2*min(draw_crit(1,:));...
                    1.2*max(draw_crit(1,:));...
                    1.2*min(draw_area(1,:));...
                    1.2*max(draw_area(1,:));...
                    X_lims_add]);
    Y_lims = sort([.7*R_excl*cos(alpha_drift_rad);...
                   -1.3*(sqrt((R_excl+L_area)^2+(W_area/2)^2))...
                                                   *cos(alpha_drift_rad);...
                   -0.8*R_excl;...
                    0.8*R_excl;...
                    1.2*min(draw_crit(2,:));...
                    1.2*max(draw_crit(2,:));...
                    1.2*min(draw_area(2,:));...
                    1.2*max(draw_area(2,:));...
                    Y_lims_add]);
    X_lim = [round(X_lims(1)) round(X_lims(end))];
    Y_lim = [round(Y_lims(1)) round(Y_lims(end))];
end;

% Legend position
if draw_labels,
    if (abs(alpha_drift)<20)    ||(abs(alpha_drift-180)<20)||...
            (abs(alpha_drift-90)<20)||(abs(alpha_drift-270)<10)||  ...
            (abs(alpha_drift-360)<20),
        loc_leg = 'BestOutside';
    elseif (abs(alpha_drift-35)<=15)||(abs(alpha_drift-245)<=15),
        loc_leg = 'SouthEast';
    elseif (abs(alpha_drift-60)<=10)||(abs(alpha_drift-215)<=15),
        loc_leg = 'NorthWest';
    elseif (abs(alpha_drift-120)<=15),
        loc_leg = 'SouthWest';
    elseif (abs(alpha_drift-150)<=15)||(abs(alpha_drift-295)<=15),
        loc_leg = 'NorthEast';
    elseif (abs(alpha_drift-325)<=15),
        loc_leg = 'SouthWest';
    else
        loc_leg = 'Best';
    end;
end;



% SIMULATION
if frame_freq ~= 0,
    if frame_freq ~= -1,
        disp('Simulation')
    else
        disp('Representation')
    end;
    fig_height = 800;
    fig_width  = fig_height*(X_lim(2)-X_lim(1))/(Y_lim(2)-Y_lim(1));
    if length(alpha_drift_rad_t) ~= 1,
        % If varying drift angle, keeping space for drift angle diagram
        fig_width  = fig_width + 400;
    end;

    if strcmp(loc_leg,'BestOutside'),
        fig_width = fig_width+400;
    end;
    figure('Position', [100 100 fig_width fig_height], 'Color',[1 1 1]),
end;

if frame_freq == -1,
    k_end = 1;
else
    k_end = floor(n_traj*length(traj_0_t));
end
if mov{1},
    fm = 1; %Frame counter
end;
ice_ch_1 = zeros(2,k_end);
ice_ch_2 = zeros(2,k_end);
psi_ship = 0; % initialization
ch = true;
time = 0;

% If constant drift angle
if length(alpha_deg_t) == 1,
    alpha_drift_rad_t = alpha_drift_rad*ones(1, k_end);
end;

for k = 1:k_end,

    %Rotation matrix
    Rot_alpha = [cos(alpha_drift_rad_t(k)) -sin(alpha_drift_rad_t(k));...
                 sin(alpha_drift_rad_t(k))  cos(alpha_drift_rad_t(k))];

    %Rotating the trajectory
    traj_t = zeros(size(traj_0_t));
    for l=1:length(traj_0_t(1,:)),
        traj_t(:,l) = traj_0_t(:,l)'*Rot_alpha;
    end;
    traj_t(1,:) = traj_t(1,:) - dist_0*sin(alpha_drift_rad_t(k));
    traj_t(2,:) = traj_t(2,:) - dist_0*cos(alpha_drift_rad_t(k));

    %Rotating the channel border
    draw_channel = zeros(2,length(draw_channel_0(1,:)));
    for l=1:length(draw_channel_0(1,:)),
        draw_channel(:,l) = draw_channel_0(:,l)'*Rot_alpha;
    end;


    time = time + dt;
    k_mod = mod(k,length(traj_t));
    if k_mod == 0, k_mod = length(traj_t); end;

    % Ship orientation
    psi_ship_old = psi_ship;
    if k_mod > 1 && k_mod < length(traj_t),
        psi_ship = atan((traj_t(2,k_mod+1)-traj_t(2,k_mod-1))...
                        /(traj_t(1,k_mod+1)-traj_t(1,k_mod-1)));
    elseif k_mod == 1,
        psi_ship = atan((traj_t(2,k_mod+1)-traj_t(2,k_mod))...
                        /(traj_t(1,k_mod+1)-traj_t(1,k_mod)));
    elseif k_mod == length(traj_t),
        psi_ship = atan((traj_t(2,2)-traj_t(2,k_mod))...
                        /(traj_t(1,2)-traj_t(1,k_mod)));
    end
    % Check when psi_ship jumps from -180deg to +180deg
    if abs(psi_ship - psi_ship_old) >= pi/4, % no jump for psi_ship
        ch = ~ch;
    end
    % Broken channel calculation
    if ch,
        ice_ch_2(:,k) = [traj_t(1,k_mod)+(W_channel/2)*sin(psi_ship),...
             traj_t(2,k_mod)-(W_channel/2)*cos(psi_ship)];
        ice_ch_1(:,k) = [traj_t(1,k_mod)-(W_channel/2)*sin(psi_ship),...
             traj_t(2,k_mod)+(W_channel/2)*cos(psi_ship)];
    else
        ice_ch_1(:,k) = [traj_t(1,k_mod)+(W_channel/2)*sin(psi_ship),...
             traj_t(2,k_mod)-(W_channel/2)*cos(psi_ship)];
        ice_ch_2(:,k) = [traj_t(1,k_mod)-(W_channel/2)*sin(psi_ship),...
             traj_t(2,k_mod)+(W_channel/2)*cos(psi_ship)];
    end;

    if frame_freq ~= 0,
        % FRAME
        if length(alpha_deg_t) ~= 1,
                % If varying drift angle, drift angle diagram
                subplot(5,10,sort([1:10:41  2:10:42  3:10:43  4:10:44  ...
                                   5:10:45  6:10:46  7:10:47]))
        end;
        % Frame every frame_freq of k
        if (mod(k,frame_freq)==0) || (k==1) || (k==k_end),
            % Counter
            if frame_freq ~= -1,
                disp([num2str(k), '/', num2str(k_end)])
            end;
            %Broken channel
            if frame_freq ~= -1,
                p=1;
                q=1+frame_freq;
                for j=1:length(ice_ch_1(1,1:k)),
                    if q <= length(ice_ch_1(1,1:k)),
                        x = [ice_ch_1(1,p:q) ice_ch_2(1,q:-1:p)];
                        y = [ice_ch_1(2,p:q) ice_ch_2(2,q:-1:p)];
                    else
                        x = [ice_ch_1(1,p:k) ice_ch_2(1,k:-1:p)];
                        y = [ice_ch_1(2,p:k) ice_ch_2(2,k:-1:p)];
                    end;
                    if j==1,
                        h(6) = fill(x, y, 'y', 'EdgeColor','y');
                    else
                        fill(x, y, 'y', 'EdgeColor','y');
                    end;
                    hold on,
                    p = p+frame_freq;
                    q = q+frame_freq;
                end;
            end;

            plot(ice_ch_1(1,1:k), ice_ch_1(2,1:k), '-k',...
                 'MarkerFaceColor','k', 'MarkerSize',1)
            hold on,
            plot(ice_ch_2(1,1:k), ice_ch_2(2,1:k), '-k',...
                 'MarkerFaceColor','k', 'MarkerSize',1)
            hold on,

            % GEOMETRICAL ELEMENTS
```

```
            h(1) = fill(draw_struct(1,:), draw_struct(2,:), 'w',...
                'EdgeColor','b','LineWidth', 2);
            hold on,
            h(2) = plot(draw_crit(1,:),draw_crit(2,:),'--b',...
                'LineWidth',2);
            hold on,
            h(3) = plot(draw_excl_zone(1,:), draw_excl_zone(2,:), '--k');
            hold on,
            h(4) = plot(draw_channel(1,:), draw_channel(2,:), 'g',...
                'LineWidth', 2);
            if frame_freq == -1,
                h(6) = plot(draw_area(1,:), draw_area(2,:), 'g',...
                    'LineWidth', 4);
                hold on,
            end;
            h(5) = plot(traj_t(1,:), traj_t(2,:), 'r');
            hold on
            plot(traj_t(1,k_mod), traj_t(2,k_mod), 'sb')
            hold on,
            if (D_struct == 0) && (L_struct ~= 0) && (B_struct ~= 0),
                % If structure is rectangular, draw orientation
                quiver([0 3*X_lim(2)],[0 3*Y_lim(2)],...
                    0.3*L_struct*sin(Psi_struct_rad)*[1 30],...
                    0.3*L_struct*cos(Psi_struct_rad)*[1 30],...
                    'b', 'LineWidth', 2)
            end;

            quiver(.75*R_excl*sin(alpha_drift_rad_t(k))*[5 -1 -3],...
                .75*R_excl*cos(alpha_drift_rad_t(k))*[5 -1 -3],...
                0.5*L_struct*sin(alpha_drift_rad_t(k))*[1 .1 .1],...
                0.5*L_struct*cos(alpha_drift_rad_t(k))*[1 .1 .1],...
                'g', 'LineWidth', 2)

            if draw_labels,
                if frame_freq ~= -1,
                    h_leg = legend(h,{'Structure', 'Critical zone',...
                        'Ships exclusion zone',...
                        'Working channel',...
                        'Vessel trajectory',...
                        'Broken channel'},...
                        'Orientation', 'Vertical');
                else
                    h_leg = legend(h,{'Structure', 'Critical zone',...
                        'Ships exclusion zone',...
                        'Working channel',...
                        'Vessel trajectory',...
                        'Working area'},...
                        'Orientation', 'Vertical');
                end;
                set(h_leg, 'FontSize', 7, 'Location',loc_leg)
                xlabel('$Latitude$ $[m]$', 'Fontsize',12,'Interpreter',...
                    'latex')
                ylabel('$Longitude$ $[m]$','Fontsize',12,'Interpreter',...
                    'latex')
                if frame_freq ~= -1,
                    title(['Simulation of Ice management operations ',...
                        '- Reference : ', Ref], 'Fontsize', 12)
                else
                    title(['Ice management configuration - ',...
                        'Reference : ', Ref], 'Fontsize', 12)
                end;


                % Display time
                mTextBox = uicontrol('style','text');
                set(mTextBox,'Position',...
                                [fig_width/2-30 fig_height/260 100 18])
                set(mTextBox,'BackgroundColor',[1 1 1])
                if time < 60,
                    set(mTextBox,'String',['t = ', num2str(time),' s'])
                else
                    set(mTextBox,'String',['t = ',...
                        num2str(floor(time/60)),' min'])
                end


            end;
            axis([X_lim Y_lim])
            daspect([1 1 1])
            hold off,

            if length(alpha_deg_t) ~= 1,
                % If varying drift angle, drift angle diagram
                subplot(5,10,sort([19:10:39 20:10:40]))
                plot((1:length(alpha_deg_t))*dt,alpha_deg_t)
                hold on,
                plot(k*dt,alpha_deg_t(k), 'd', 'MarkerSize', 12)
                xlabel('$time$ $[s]$', 'Fontsize',12, 'Interpreter',...
                    'latex')
                ylabel('$\alpha_{drift}$ $[deg]$', 'Fontsize',12,...
                    'Interpreter', 'latex')
            end

            if mov{1},
                frame_s(fm) = getframe(gcf);
                fm = fm + 1;
            else
                getframe;
            end;
            hold off,
        end;
    end;

    % ICE DRIFT
    ice_ch_1(1,:) = ice_ch_1(1,:) + V_drift*sin(alpha_drift_rad_t(k))*dt;
    ice_ch_1(2,:) = ice_ch_1(2,:) + V_drift*cos(alpha_drift_rad_t(k))*dt;
    ice_ch_2(1,:) = ice_ch_2(1,:) + V_drift*sin(alpha_drift_rad_t(k))*dt;
    ice_ch_2(2,:) = ice_ch_2(2,:) + V_drift*cos(alpha_drift_rad_t(k))*dt;

end;
if mov{1},
    disp('Recording movie ...')
    movie2avi(frame_s, ['Movies\',mov{2},'.avi'], 'FPS', mov{3})
    disp('Done.')
end
```

## D.8 Resulting Floe Size Analysis

### *D.8.1 Floe_size_analysis_linear.m*

```
function G = Floe_size_analysis_linear(ice_ch_1, ice_ch_2, N_pass,...
                W_channel, t_cycle, fig_local, fig_main)
%
% Analysis of the floe size distribution after ice management, considering
% linear technique.
%
% INPUTS :
% ice_ch_1      [m]   Coordinates (x,y) of the broken channel edge 1
% ice_ch_2      [m]   Coordinates (x,y) of the broken channel edge 2
% N_pass        [-]   Number of passes of the ice management trajectory
% W_channel     [m]     Icebreaker channel width
% t_cycle       [s]   Cycle time of the ice management trajectory
% fig_local       [Bool] Activate/Deactivate plotting of sampling area
% fig_main        [Bool] Activate/Deactivate plotting of results histograms
%
% OUTPUT:
% G               [m^2]  Vector of broken floes areas
%

global Ice_Water_data
global Structure_data
global Ice_drift_data
global Other_data


% READING ICE WATER PARAMETERS % CHANGED
H_ice      = Ice_Water_data{2};      % Ice thickness
H_snow     = Ice_Water_data{3}(1);  % Snow thickness
rho_i      = Ice_Water_data{5}(2);  % Ice density
rho_s      = Ice_Water_data{5}(3);  % Snow density


% READING ICE DRIFT DATA
V_drift          = Ice_drift_data{2};  % Drift velocity
alpha_drift      = Ice_drift_data{3};  % Drift direction
alpha_drift_rad = alpha_drift*pi/180;


% READING STRUCTURE PARAMETERS
Ref              = Structure_data{1};    % Reference
d_floe_target    = Structure_data{6};    % Target floe size


% READING OTHER PARAMETERS
dt               = Other_data{3};    % Time step
dx_ice           = Other_data{4};    % Spatial step



disp('FLOE SIZE ANALYSIS')

disp(' - Sampling of the trajectory')

% Rotation matrix for alpha_drift
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];

% Rotation of the broken channel parallel to the vertical axis
ice_ch_1_par = zeros(2, length(ice_ch_1));
ice_ch_2_par = zeros(2, length(ice_ch_1));

for k=1:length(ice_ch_1),
    ice_ch_1_par(:,k) = ice_ch_1(:,k)'/Rot_alpha;
    ice_ch_2_par(:,k) = ice_ch_2(:,k)'/Rot_alpha;
end;

% Width of the actual managed area
% (considering the number of passes, instead of the structure dimensions)
W_area_eff = N_pass*W_channel + (N_pass-1)*d_floe_target;

% Starting point of the floe analysis
t_start = 1*t_cycle;
k_start = round(t_start/dt);

% End point of the floe analysis
n_cycle = floor(length(ice_ch_1)/(t_cycle/dt));%Number of completed cycles
if n_cycle >= 3,
  t_end = (n_cycle-1)*t_cycle;
  k_end = round(t_end/dt);
else
  error(['Unsufficient number of completed management cycles. ',...
         'Please run at least 3 entire cycles.'])
end;

disp(' - Calculation of broken ice areas')

% Length of analysed channel area
L_ch_analysis =   max([ice_ch_1_par(2, k_start:k_end)...
                       ice_ch_2_par(2, k_start:k_end)])...
            - min([ice_ch_1_par(2, k_start:k_end)...
                       ice_ch_2_par(2, k_start:k_end)]);

% Ice matrix coordinates
DX  = (0:(floor(W_area_eff/dx_ice)+1))*dx_ice - W_area_eff/2;
DY  = (0:(floor(L_ch_analysis/dx_ice)+1))*dx_ice +...
      min([ice_ch_1_par(2, k_start:k_end) ice_ch_2_par(2, k_start:k_end)]);


% use of poly2msak, or inpolygon ?
kx_1_ice = zeros(1, length(ice_ch_1_par));
ky_1_ice = zeros(1, length(ice_ch_1_par));
kx_2_ice = zeros(1, length(ice_ch_1_par));
ky_2_ice = zeros(1, length(ice_ch_1_par));
for k = 1:length(ice_ch_1_par),
    % Current points
    x_1 = ice_ch_1_par(1,k);
    y_1 = ice_ch_1_par(2,k);
    x_2 = ice_ch_2_par(1,k);
    y_2 = ice_ch_2_par(2,k);

    % Closest points in the ICE matrix
    [x_1_min, kx_1_ice(k)] = min(abs(DX-x_1));
    [y_1_min, ky_1_ice(k)] = min(abs(DY-y_1));
    [x_2_min, kx_2_ice(k)] = min(abs(DX-x_2));
    [y_2_min, ky_2_ice(k)] = min(abs(DY-y_2));
end;

% Broken channel in ICE matrix
p = 1;
q = 1 + round(t_cycle/128);
ICE = false(floor(L_ch_analysis/dx_ice)+1, floor(W_area_eff/dx_ice)+1);
while q < length(kx_1_ice),
    kx_ch_par_pq = [kx_1_ice(p:q) kx_2_ice(q:-1:p)];
    ky_ch_par_pq = [ky_1_ice(p:q) ky_2_ice(q:-1:p)];
    ICE_add = poly2mask(kx_ch_par_pq, ky_ch_par_pq,...
              floor(L_ch_analysis/dx_ice)+1, floor(W_area_eff/dx_ice)+1);
    ICE = ICE | ICE_add;
    p = p + round(t_cycle/128);
    q = q + round(t_cycle/128);
end;
ICE = ~ICE(2:end, 2:end);



% Pre-cuts along the linear parts of the trajectory
% (it is assumed that the cracks between two linear parts of successive
% cycles will propagate)
x_pass_par  = zeros(1,N_pass);
kx_pass_ice = zeros(1,N_pass);
x_pass_par(1) = min(DX) + W_channel/2;
for k = 2:N_pass,
    x_pass_par(k) = x_pass_par(k-1) + W_channel + d_floe_target;
end;
for k = 1:N_pass,
    [x_pass_min, kx_pass_ice(k)] = min(abs(DX-x_pass_par(k)));
end;
ICE(:,kx_pass_ice) = false;



% Floe size detection
disp(' - Floe size detection')
F = zeros(4, 1000); % Active floe size data matrix, each column represents
                    % a different floe, starting at floe number 1.
                    % [floe number, ice piece number of the previous line,
                    %  area, bottom y-coordinate]
F_m1    = F;

lgth_1 = 100;       % Base length for initialisation
G = zeros(1,lgth_1);% Vector of broken floes areas
Links = zeros(2,lgth_1); % Matrix of Links for connections of ice pieces
                         % between lines

f = 0;              % Number of broken floes
for i = 1:(length(DY)-2), % horizontal lines
    % Looking for the ice pieces on the line
    Line = ICE(i,:);
    % Look for all the starting and ending points of ice
    k_s = zeros(1, ceil(length(Line)/2)); % max possible size
    k_e = zeros(1, ceil(length(Line)/2)); % max possible size
    n_f = 0; % number of ice parts on the line
    for j = 1:length(Line),
        if (n_f==0) && Line(j),
            % 1st piece of ice of the line
            n_f = 1;
            k_s(n_f) = j;
            if (j==length(Line)) && (Line(j)),
                %Piece of ice is one element long and touching channel
                %side
                %disp('Floe along side of managed area (case 1).')
                k_e(n_f) = j;
            end;
        elseif (n_f>=1) && (~Line(j-1)) && (Line(j)),
            % New piece of ice in the line
            n_f = n_f+1;
            k_s(n_f) = j;
            if (j==length(Line)) && (Line(j)),
                %Piece of ice is one element long and touching channel
                %side
                %disp('Floe along side of managed area (case 1).')
                k_e(n_f) = j;
            end;
        elseif (n_f>=1) && (~Line(j)) && (k_e(n_f)==0),
            % End of the current ice piece
            k_e(n_f) = j-1;
        elseif (j==length(Line)) && (Line(j)),
            % The last ice piece is touching the channel side
            % We assume this ice piece as a brohen floe (hypothesis)
            % disp('Floe along side of managed area (case 2).')
            k_e(n_f) = j;
        %elseif (n_f>=1) && (Line(j)) && (k_e(n_f)~=0),
            % We are in the ice piece
        %elseif (n_f>=1) && (~Line(j)) && (k_e(n_f)~=0),
            % We are in the broken channel
        end;
    end;

    Links(:,:) = 0;
    l = 1;
    if i > 1,
        % We try to associate with the ice pieces of the previous line
        for n = 1:n_f,
            for m = 1:n_f_im1,
                if (k_s_im1(m) >= k_s(n)) && (k_s_im1(m) <= k_e(n))||  ...
                   (k_e_im1(m) >= k_s(n)) && (k_e_im1(m) <= k_e(n))||  ...
                   (k_s(n) >= k_s_im1(m)) && (k_s(n) <= k_e_im1(m))||  ...
                   (k_e(n) >= k_s_im1(m)) && (k_e(n) <= k_e_im1(m)),
                    Links(:,l) = [n; m];
                    l = l+1;
                end;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            end;
            % If no link found for ice piece n, an empty link is created,
            % so that a new floe will be detected.
            if (l > 1) && (Links(1,l-1) == n-1),
                Links(:,l) = [n; 0];
                l = l+1;
            %   disp('Empty link')
            elseif (l == 1) && (Links(1,l) == 0),
                Links(:,l) = [n; 0];
                l = l+1;
            %   disp('Empty link for 1st cell')
            end;
        end;
    end;
    %f_del = 0; %Number of deleted floes in this line
    % Creation or extension of the floes according to the Links matrix
    if i == 1, % Only new floes are created
        for n = 1:n_f,
            %Floe number, ice piece in line i-1, area; bottom y-coordinate
            F(:,n) = [n; n; (k_e(n)-k_s(n)+1)*dx_ice^2; (i-1)*dx_ice];
        end;
        f_e = n_f; % Largest active floe number (last registered floe nbr)
```

```
        f_e_c = n_f;% Column index of the largest active floe number
    else
        F2 = F(2,:);
        for ll=1:(l-1),
            F_2p = F(2,:);
            if Links(2,ll) ~= 0,
                p = 1;
                while (F_2p(p) ~= Links(2,ll)) && (p < length(F_2p)),
                    p = p+1;
                end;
                %if p >= length(F_2p),
                %    disp('Skipped ice piece')
                %end;

                % IF statement to check for ice pieces linked to several
                % floes
                if (ll==1) || ((ll>=2) && (Links(1,ll-1)~=Links(1,ll))),
                    % IF statement to check for consecutive ice pieces
                    % linked to the same floe
                    if (ll==1) || ((ll>=2)&&(Links(2,ll-1)~=Links(2,ll))),
                        F2(p) = Links(1,ll);
                        F(3,p) = F(3,p)...
                          +(k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;
                    else
                        % If two consecutive ice pieces for the same floe,
                        % no change and we keep the lowest number.
                        %disp('Consecutive ice pieces for the same floe')
                        lll = ll;
                        % Looking for the 1st ice piece of the series
                        while (lll > 1) && (Links(2,lll-1)==Links(2,lll)),
                            lll = lll - 1;
                        end;
                        F(3,p) = F(3,p)...
                          +(k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;
                        k_e(Links(1,lll)) = k_e(Links(1,ll));
                        k_e(Links(1,ll))=length(Line)+1;%Out of the domain
                        k_s(Links(1,ll))=length(Line)+1;%Out of the domain
                    end;
                else
                    % The current ice piece connects to two initially
                    % independent floes
                    % => the area of the two floes are combined into the
                    % 1st one.
                    % The residual floe number is set at zero and will
                    % stay in F until the end of the calculation
                    % disp(['Current ice piece connects to two ',...
                    %       'initially independent floes'])
                    F(1,p) = 0;

                    pp = 1;
                    while F_2p(pp) ~= F(2,p)-1,
                        pp = pp+1;
                    end;

                    F(3,pp) = F(3,pp) + F(3,p);
                    F(:,p) = [];
                    F2(p)  = [];
                    f_e_c = f_e_c - 1;
                end;
            else %New floe
                %disp('New floe')
                f_e = f_e + 1;
                f_e_c = f_e_c + 1;
                %Floe number; ice piece in line i-1; area; bottom y-coord.
                F(:,f_e_c) = [f_e;...
                    Links(1,ll);...
                    (k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;...
                    (i-1)*dx_ice];
                F2(f_e_c) = Links(1,ll);
            end;
        end;
        F(2,:) = F2;
    end;

    % Check for floe breaking
    y = (i-1)*dx_ice; % y-coordinate of the current line
    q = 1;
      pp = zeros(1, f_e_c);
    for p = 1:f_e_c,
      if ((y - F(4,p) + dx_ice) >= d_floe_target) && (F(1,p) ~= 0),
              %Floe breaks because of its size
              %disp(['Floe breaking (number ', num2str(F(1,p)), ')'])
        pp(q) = p;
        q     = q+1;
              k_s(F(2,p)) = 0;
              k_e(F(2,p)) = 0;
        elseif (F(3,p) == F_m1(3,p)) && (F(1,p) ~= 0),
            %Floe has not changed of area, meaning that it is isolated and
            %cannot grow anymore
            %disp(['Isolated floe (number ', num2str(F(1,p)), ')'])
        pp(q) = p;
        q     = q+1;
      end;
    end;
      pp(q:end) = [];
      if f+length(pp) > length(G),
          % Increasing size of G
          G = horzcat(G,zeros(1,lgth_1));
      end;
    G((f+1):(f+length(pp)))=F(3,pp);%We keep only the floe area in memory
    F(:,pp) = [];
      F       = horzcat(F,zeros(4,length(pp)));%Keeping constant size for F
      f_e_c   = f_e_c - length(pp);
    f         = f + length(pp);

    % We remember the previous line for the calculation of next line
    k_s_im1 = k_s;
    k_e_im1 = k_e;
    n_f_im1 = n_f;
      F_m1    = F;
end;

G((f+1):end) = [];

% OUTPUTS
disp(' - Floe size analysis :')
disp(['   + Calculated broken floes : ', num2str(f)])
disp(['   + Mean floe size: ', num2str(mean(G),5),' m^2.'])
disp(['   + Biggest : ', num2str(max(G)),' m^2.'])
disp(['   + Smallest : ', num2str(min(G)),' m^2.'])


% FIGURES
if fig_main,
    if not(fig_local),
        figure('Position', [0 0 600 450]),
        nbins = 30;

        subplot(10,1,1:7)
        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
            'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main  = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', mat2cell(X_tick_mass))

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                          *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', mat2cell(X_tick_momentum))

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')
    else
        figure('Position', [0 0 1000 450]),
        nbins = 30;

        subplot(10,6,[1:3 7:9 13:15 19:21 25:27 31:32 37:38])

        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
                'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main  = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', {X_tick_mass})

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                          *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', {X_tick_momentum})

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')

        % Figure of the sampling area
        subplot(10,6,...
              [5:6 11:12 17:18 23:24 29:30 35:36 41:42 47:48 53:54 59:60])

        imagesc(DX, DY, ICE)
        set(gca,'YDir','normal')
        colormap(gray)
        daspect([1 1 1])
        title('Sampling area')
        xlabel('\textit{x} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
        ylabel('\textit{y} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
    end;
end;
end
```

### *D.8.2 Floe_size_analysis_sector.m*

```
function G = Floe_size_analysis_sector(ice_ch_1, ice_ch_2, t_cycle,...
                 e_h, fig_local, fig_main)
%
% Analysis of the floe size distribution after ice management, considering
% sector technique.
%
% INPUTS :
% ice_ch_1      [m]   Coordinates (x,y) of the broken channel edge 1
% ice_ch_2      [m]   Coordinates (x,y) of the broken channel edge 2
% t_cycle       [s]   Cycle time of the ice management trajectory
% fig_local       [Bool] Activate/Deactivate plotting of sampling area
% fig_main        [Bool] Activate/Deactivate plotting of results histograms
%
% OUTPUT :
% G               [m^2]  Vector of broken floes areas
%

global Ice_Water_data
global Structure_data
global Ice_drift_data
global Other_data

% READING ICE WATER PARAMETERS % CHANGED
H_ice    = Ice_Water_data{2};      % Ice thickness
H_snow   = Ice_Water_data{3}(1);  % Snow thickness
rho_i    = Ice_Water_data{5}(2);  % Ice density
rho_s    = Ice_Water_data{5}(3);  % Snow density


% READING ICE DRIFT DATA
V_drift         = Ice_drift_data{2};  % Drift velocity
alpha_drift     = Ice_drift_data{3};  % Drift direction
alpha_drift_rad = alpha_drift*pi/180;


% READING STRUCTURE PARAMETERS
Ref             = Structure_data{1};    % Reference
d_floe_target   = Structure_data{6};    % Target floe size

% READING OTHER PARAMETERS
dt              = Other_data{3};    % Time step
dx_ice          = Other_data{4};    % Spatial step

disp('FLOE SIZE ANALYSIS')

disp(' - Sampling of the trajectory')

% Rotation matrix for alpha_drift
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];

% Rotation of the broken channel parallel to the vertical axis
ice_ch_1_par = zeros(2, length(ice_ch_1));
ice_ch_2_par = zeros(2, length(ice_ch_1));

for k=1:length(ice_ch_1),
    ice_ch_1_par(:,k) = ice_ch_1(:,k)'/Rot_alpha;
    ice_ch_2_par(:,k) = ice_ch_2(:,k)'/Rot_alpha;
end;

% Width of the actual managed area (measured from computed managed area)
W_area_act =    max([ice_ch_1_par(1,:) ice_ch_2_par(1,:)])...
              - min([ice_ch_1_par(1,:) ice_ch_2_par(1,:)]);



% Starting point of the floe analysis
t_start = 1*t_cycle;
k_start = round(t_start/dt);

% End point of the floe analysis
n_cycle =floor(length(ice_ch_1)/(t_cycle/dt));%Completed management cycles
if n_cycle >= 3,
  t_end = (n_cycle-1)*t_cycle;
  k_end = round(t_end/dt);
else
  error(['Unsufficient number of completed management cycles. ',...
         'Please run at least 3 entire cycles.'])
end;

% Length of analysed channel area
L_ch_analysis =   max([ice_ch_1_par(2, k_start:k_end)...
                       ice_ch_2_par(2, k_start:k_end)])...
                - min([ice_ch_1_par(2, k_start:k_end)...
                       ice_ch_2_par(2, k_start:k_end)]);

% Ice matrix coordinates
DX  = (0:(floor(W_area_act/dx_ice)+1))*dx_ice - W_area_act/2 + e_h;
DY  = (0:(floor(L_ch_analysis/dx_ice)+1))*dx_ice +...
      min([ice_ch_1_par(2,k_start:k_end) ice_ch_2_par(2,k_start:k_end)]);

% use of poly2msak, or inpolygon ?
kx_1_ice = zeros(1, length(ice_ch_1_par));
ky_1_ice = zeros(1, length(ice_ch_1_par));
kx_2_ice = zeros(1, length(ice_ch_1_par));
ky_2_ice = zeros(1, length(ice_ch_1_par));
for k = 1:length(ice_ch_1_par),
    % Current points
    x_1 = ice_ch_1_par(1,k);
    y_1 = ice_ch_1_par(2,k);
    x_2 = ice_ch_2_par(1,k);
    y_2 = ice_ch_2_par(2,k);

    % Closest points in the ICE matrix
    [x_1_min, kx_1_ice(k)] = min(abs(DX-x_1));
    [y_1_min, ky_1_ice(k)] = min(abs(DY-y_1));
    [x_2_min, kx_2_ice(k)] = min(abs(DX-x_2));
    [y_2_min, ky_2_ice(k)] = min(abs(DY-y_2));
end;

% Broken channel in ICE matrix
p = 1;
q = 1 + round(t_cycle/16);
ICE = false(floor(L_ch_analysis/dx_ice)+1, floor(W_area_act/dx_ice)+1);
while q < length(kx_1_ice),
    kx_ch_par_pq = [kx_1_ice(p:q) kx_2_ice(q:-1:p)];
    ky_ch_par_pq = [ky_1_ice(p:q) ky_2_ice(q:-1:p)];
    ICE_add = poly2mask(kx_ch_par_pq, ky_ch_par_pq,...
                floor(L_ch_analysis/dx_ice)+1, floor(W_area_act/dx_ice)+1);
    ICE = ICE | ICE_add;
    p = p + round(t_cycle/16);
    q = q + round(t_cycle/16);
end;
ICE = ~ICE(2:end, 2:end);

% Removing the ice parts in the below and top of the matrix, as they are
% cut by the sampling area, and therefore are not realistic
for i = 1:(length(DX)-2), % vertical lines
    j=1;
    while (j < (length(DY)-2)) && ICE(j,i),
        ICE(j,i) = false;
        j = j+1;
    end;

    j=(length(DY)-2);
    while (j > 0) && ICE(j,i),
        ICE(j,i) = false;
        j = j-1;
    end;
end;


% Floe size detection
disp(' - Floe size detection')

lgth_1 = 100;        % Base length for initialisation
F = zeros(3,lgth_1);% Active floe size data matrix, each column represents
                    % a different floe, starting at floe number 1.
                    % [floe number, ice piece number of the previous line,
                    %  left x-coordinate]
F_a = zeros(length(DX)-1,lgth_1);% Floe area matrix, each column
                    % represents a different floe, and the lines contain
                    % the floe area at each x-coordinate

f_e = 0;  % Largest active floe number (last registered floe nbr)
for i = 1:(length(DX)-2), % vertical lines

    if f_e > (length(F(1,:))-50),
        % Increasing size of G
        F   = horzcat(F,zeros(3,lgth_1));
        F_a = horzcat(F_a,zeros(length(DX)-1,lgth_1));
    end;

    % Looking for the ice pieces on the vertical
    Vertic = ICE(:,i);
    % Look for all the starting and ending points of ice
    k_s = zeros(1, ceil(length(Vertic)/2)); % max possible size
    k_e = zeros(1, ceil(length(Vertic)/2)); % max possible size
    n_f = 0; % number of ice parts on the vertical
    for j = 1:length(Vertic),
        if (n_f==0) && Vertic(j),
            % 1st piece of ice of the vertical
            n_f = 1;
            k_s(n_f) = j;
        elseif (n_f>=1) && (~Vertic(j-1)) && (Vertic(j)),
            % New piece of ice in the vertical
            n_f = n_f+1;
            k_s(n_f) = j;
            if (j==length(Vertic)) && (Vertic(j)),
                %Piece of ice is one element long and touching channel
                %side
                %disp('Floe along side of managed area (case 1).')
                k_e(n_f) = j;
            end;
        elseif (n_f>=1) && (~Vertic(j)) && (k_e(n_f)==0),
            % End of the current ice piece
            k_e(n_f) = j-1;
        elseif (j==length(Vertic)) && (Vertic(j)),
            % The last ice piece is touching the channel side
            % We assume this ice piece as a brohen floe (hypothesis)
            % disp('Floe along side of managed area (case 2).')
            k_e(n_f) = j;
        %elseif (n_f>=1) && (Vertic(j)) && (k_e(n_f)~=0),
            % We are in the ice piece
        %elseif (n_f>=1) && (~Vertic(j)) && (k_e(n_f)~=0),
            % We are in the broken channel
        end;
    end;


    l = 1;
    Links = zeros(2,lgth_1); % Matrix of Links for connections of ice
                             % pieces between lines
    if i > 1,
        % We try to associate with the ice pieces of the previous vertical
        for n = 1:n_f,
            for m = 1:n_f_im1,
                if (k_s_im1(m) >= k_s(n)) && (k_s_im1(m) <= k_e(n))||  ...
                   (k_e_im1(m) >= k_s(n)) && (k_e_im1(m) <= k_e(n))||  ...
                   (k_s(n) >= k_s_im1(m)) && (k_s(n) <= k_e_im1(m))||  ...
                   (k_e(n) >= k_s_im1(m)) && (k_e(n) <= k_e_im1(m)),
                    Links(:,l) = [n; m];
                    l = l+1;
                end;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            end;
            % If no link found for ice piece n, an empty link is created,
            % so that a new floe will be detected.
            if (l > 1) && (Links(1,l-1) == n-1),
                %   disp('Empty link')
                Links(:,l) = [n; 0];
                l = l+1;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            elseif (l == 1) && (Links(1,l) == 0),
                %   disp('Empty link for 1st cell')
                Links(:,l) = [n; 0];
                l = l+1;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            end;
        end;
    end;

    % Creation or extension of the floes according to the Links matrix
    if i == 1, % Only new floes are created
        for n = 1:n_f,
            % Floe number, ice piece in line i-1, left x-coordinate
            F(:,n)   = [n; n; (i-1)*dx_ice];
            % Area related to that floe on the vertical
            F_a(i,n) = (k_e(n)-k_s(n)+1)*dx_ice^2;
        end;
        f_e = n_f; % Largest active floe number (last registered floe nbr)
    else
        F_2p = zeros(1,length(F(2,:)));
        for ll=1:(l-1),
            if Links(2,ll) ~= 0,
```

```
                p = 1;
                while (F(2,p) ~= Links(2,ll)) && (p < length(F(2,:))),
                    p = p+1;
                end;
                F_2p(p) = Links(1,ll);
                F_a(i,p) = (k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;


            else %New floe
                %disp('New floe')
                f_e = f_e + 1;

                F_2p(f_e) = Links(1,ll);
                % Floe number, ice piece in line i-1, left x-coordinate
                F([1 3],f_e)   = [f_e; (i-1)*dx_ice];
                % Area related to that floe on the vertical
                F_a(i,f_e) = ...
                            (k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;
            end;
        end;
        F(2,:) = F_2p;
    end;

    % We remember the previous line for the calculation of next line
  k_s_im1 = k_s;
  k_e_im1 = k_e;
  n_f_im1 = n_f;
end;

F(:,(f_e+1):end) = [];
F_a(:,(f_e+1):end) = [];

% Calculation of the floes areas
G = zeros(1,F(1,end)*(length(DX)-2)); % Vector of broken floes areas
                                      % (Initialized to its maximum
                                      % possible size)
g=1;
for f=1:F(1,end),
    p_s=F(3,f)/dx_ice+1; % Index of the 1st left element of ice of the floe
    p_x=p_s;
    while F_a(p_x,f) ~=0,
        floe_lgth = (p_x-p_s+1)*dx_ice;
        if floe_lgth < d_floe_target,
            G(g) = G(g) + F_a(p_x,f); % Size of current floe increases
        else %floe breaks
            %disp(['Floe breaks in x = ', num2str(p_x*dx_ice), 'm.'])
            g = g+1;
            G(g)= F_a(p_x,f); %New floe
            p_s = p_x;
        end;
        p_x=p_x+1;
    end;
    g = g+1;
end;
G(g:end) = [];


% OUTPUTS
disp(' - Floe size analysis :')
disp(['   + Calculated broken floes : ', num2str(g-1)])
disp(['   + Mean floe size: ', num2str(mean(G),5),' m^2.'])
disp(['   + Biggest : ', num2str(max(G)),' m^2.'])
disp(['   + Smallest : ', num2str(min(G)),' m^2.'])

% FIGURES
if fig_main,
    if not(fig_local),
        figure('Position', [0 0 600 450]),
        nbins = 30;

        subplot(10,1,1:7)

        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
            'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', {X_tick_mass})

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                        *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', {X_tick_momentum})

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')
    else
        figure('Position', [0 0 1000 450]),
        nbins = 30;

        subplot(10,6,[1:3 7:9 13:15 19:21 25:27 31:32 37:38])

        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
                'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', {X_tick_mass})

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                        *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', {X_tick_momentum})

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')

        % Figure of the sampling area
        subplot(10,6,...
                [5:6 11:12 17:18 23:24 29:30 35:36 41:42 47:48 53:54 59:60])

        imagesc(DX, DY, ICE)
        set(gca,'YDir','normal')
        colormap(gray)
        daspect([1 1 1])
        title('Sampling area')
        xlabel('\textit{x} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
        ylabel('\textit{y} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
    end;
end;
end
```

### *D.8.3 Floe_size_analysis_circular.m*

```
function G = Floe_size_analysis_circular(ice_ch_1, ice_ch_2, t_cycle,...
                e_h, R_traj, fig_local, fig_main)
%
% Analysis of the floe size distribution after ice management, considering
% circular technique.
%
% INPUTS :
% ice_ch_1      [m]   Coordinates (x,y) of the broken channel edge 1
% ice_ch_2      [m]   Coordinates (x,y) of the broken channel edge 2
% t_cycle       [s]   Cycle time of the ice management trajectory
% R_traj          [m]     Trajectory radius
% e_h             [m]     Offset of trajectory centre vs structure centre
% fig_local       [Bool] Activate/Deactivate plotting of sampling area
% fig_main        [Bool] Activate/Deactivate plotting of results histograms
%
% OUTPUT:
% G               [m^2]  Vector of broken floes areas
%

global Ice_Water_data
global Ship_data
global Structure_data
global Ice_drift_data
global Other_data

% READING ICE WATER PARAMETERS
H_ice    = Ice_Water_data{2};          % Ice thickness
H_snow   = Ice_Water_data{3}(1);       % Snow thickness
rho_i    = Ice_Water_data{5}(2);       % Ice density
rho_s    = Ice_Water_data{5}(3);       % Snow density


% READING ICE DRIFT DATA
V_drift          = Ice_drift_data{2};  % Drift velocity
alpha_drift      = Ice_drift_data{3};  % Drift direction
alpha_drift_rad = alpha_drift*pi/180;

% READING SHIP PARAMETERS
B_ship           = Ship_data{3};       % Ship's breadth
k_channel      = Ship_data{18};       % Channel width to ship's beam ratio


% READING STRUCTURE PARAMETERS
Ref              = Structure_data{1};  % Reference
d_floe_target    = Structure_data{6};  % Target floe size


% READING OTHER PARAMETERS
dt               = Other_data{3};      % Time step
dx_ice           = Other_data{4};     % Spatial step

disp('FLOE SIZE ANALYSIS')

disp(' - Sampling of the trajectory')

% Rotation matrix for alpha_drift
Rot_alpha = [cos(alpha_drift_rad) -sin(alpha_drift_rad);...
    sin(alpha_drift_rad)  cos(alpha_drift_rad)];

% Rotation of the broken channel parallel to the vertical axis
ice_ch_1_par = zeros(2, length(ice_ch_1));
ice_ch_2_par = zeros(2, length(ice_ch_1));

for k=1:length(ice_ch_1),
    ice_ch_1_par(:,k) = ice_ch_1(:,k)'/Rot_alpha;
    ice_ch_2_par(:,k) = ice_ch_2(:,k)'/Rot_alpha;
end;

% Width of the actual managed area (measured from computed managed area)
W_area_act =   max([ice_ch_1_par(1,:) ice_ch_2_par(1,:)])...
             - min([ice_ch_1_par(1,:) ice_ch_2_par(1,:)]);

% Icebreaker channel width
W_channel = B_ship*k_channel;

% Starting point of the floe analysis
t_start = 1*t_cycle;
k_start = round(t_start/dt);

% End point of the floe analysis
k_end = length(ice_ch_1_par(2,:));
while ice_ch_1_par(2,k_end)<(ice_ch_1_par(2,end) + 2*R_traj + W_channel),
    k_end = k_end - round(t_cycle/dt);
    if k_end < 1,
        n_cycle_min = 1+ceil((R_traj*2+V_drift*t_cycle-W_channel)...
                    /abs(ice_ch_1_par(2,t_cycle/dt)-ice_ch_1_par(2,1)));
        error(['Unsufficient number of completed management cycles. ',...
               'Please run at least ',num2str(n_cycle_min),...
               ' entire cycles.'])
    end;
end;


% Length of analysed channel area
L_ch_analysis =   max([ice_ch_1_par(2, k_start)...
                       ice_ch_1_par(2, k_end)...
                       ice_ch_2_par(2, k_start)...
                       ice_ch_2_par(2, k_end)])...
                - min([ice_ch_1_par(2, k_start)...
                       ice_ch_1_par(2, k_end)...
                       ice_ch_2_par(2, k_start)...
                       ice_ch_2_par(2, k_end)]);

% Ice matrix coordinates
DX  = (0:(floor(W_area_act/dx_ice)+1))*dx_ice - W_area_act/2 + e_h;
DY  = (0:(floor(L_ch_analysis/dx_ice)+1))*dx_ice +...
     min([ice_ch_1_par(2, k_start:k_end) ice_ch_2_par(2, k_start:k_end)]);

% use of poly2msak, or inpolygon ?
kx_1_ice = zeros(1, length(ice_ch_1_par));
ky_1_ice = zeros(1, length(ice_ch_1_par));
kx_2_ice = zeros(1, length(ice_ch_1_par));
ky_2_ice = zeros(1, length(ice_ch_1_par));
for k = 1:length(ice_ch_1_par),
    % Current points
    x_1 = ice_ch_1_par(1,k);
    y_1 = ice_ch_1_par(2,k);
    x_2 = ice_ch_2_par(1,k);
    y_2 = ice_ch_2_par(2,k);

    % Closest points in the ICE matrix
    [x_1_min, kx_1_ice(k)] = min(abs(DX-x_1));
    [y_1_min, ky_1_ice(k)] = min(abs(DY-y_1));
    [x_2_min, kx_2_ice(k)] = min(abs(DX-x_2));
    [y_2_min, ky_2_ice(k)] = min(abs(DY-y_2));
end;


% Broken channel in ICE matrix
p = 1;
q = 1 + round(t_cycle/16);
ICE = false(floor(L_ch_analysis/dx_ice)+1, floor(W_area_act/dx_ice)+1);
while q < length(kx_1_ice),
    kx_ch_par_pq = [kx_1_ice(p:q) kx_2_ice(q:-1:p)];
    ky_ch_par_pq = [ky_1_ice(p:q) ky_2_ice(q:-1:p)];
    ICE_add = poly2mask(kx_ch_par_pq, ky_ch_par_pq,...
              floor(L_ch_analysis/dx_ice)+1, floor(W_area_act/dx_ice)+1);
    ICE = ICE | ICE_add;
    p = p + round(t_cycle/16);
    q = q + round(t_cycle/16);
end;
ICE = ~ICE(2:end, 2:end);


% Floe size detection
disp(' - Floe size detection')
F = zeros(4, 1000); % Active floe size data matrix, each column represents
                    % a different floe, starting at floe number 1.
                    % [floe number, ice piece number of the previous line,
                    %  area, bottom y-coordinate]
F_m1    = F;

lgth_1 = 100;       % Base length for initialisation
G = zeros(1,lgth_1);% Vector of broken floes areas
Links = zeros(2,lgth_1); % Matrix of Links for connections of ice pieces
                         % between lines

f = 0;              % Number of broken floes
for i = 1:(length(DX)-2), % vertical lines
    % Looking for the ice pieces on the vertical
    Vertic = ICE(:,i);
    % Look for all the starting and ending points of ice
    k_s = zeros(1, ceil(length(Vertic)/2)); % max possible size
    k_e = zeros(1, ceil(length(Vertic)/2)); % max possible size
    n_f = 0; % number of ice parts on the line
    for j = 1:length(Vertic),
        if (n_f==0) && Vertic(j),
            % 1st piece of ice of the vertical
            n_f = 1;
            k_s(n_f) = j;
            if (j==length(Vertic)) && (Vertic(j)),
                %Piece of ice is one element long and touching channel
                %side
                %disp('Floe along side of managed area (case 1).')
                k_e(n_f) = j;
            end;
        elseif (n_f>=1) && (~Vertic(j-1)) && (Vertic(j)),
            % New piece of ice in the vertical
            n_f = n_f+1;
            k_s(n_f) = j;
            if (j==length(Vertic)) && (Vertic(j)),
                %Piece of ice is one element long and touching channel
                %side
                %disp('Floe along side of managed area (case 1).')
                k_e(n_f) = j;
            end;
        elseif (n_f>=1) && (~Vertic(j)) && (k_e(n_f)==0),
            % End of the current ice piece
            k_e(n_f) = j-1;
        elseif (j==length(Vertic)) && (Vertic(j)),
            % The last ice piece is touching the channel side
            % We assume this ice piece as a brohen floe (hypothesis)
            % disp('Floe along side of managed area (case 2).')
            k_e(n_f) = j;
        %elseif (n_f>=1) && (Vertic(j)) && (k_e(n_f)~=0),
            % We are in the ice piece
        %elseif (n_f>=1) && (~Vertic(j)) && (k_e(n_f)~=0),
            % We are in the broken channel
        end;
    end;

    Links(:,:) = 0;
    l = 1;
    if i > 1,
        % We try to associate with the ice pieces of the previous line
        for n = 1:n_f,
            for m = 1:n_f_im1,
                if (k_s_im1(m) >= k_s(n)) && (k_s_im1(m) <= k_e(n))|| ...
                   (k_e_im1(m) >= k_s(n)) && (k_e_im1(m) <= k_e(n))|| ...
                   (k_s(n) >= k_s_im1(m)) && (k_s(n) <= k_e_im1(m))|| ...
                   (k_e(n) >= k_s_im1(m)) && (k_e(n) <= k_e_im1(m)),
                    Links(:,l) = [n; m];
                    l = l+1;
                end;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            end;
            % If no link found for ice piece n, an empty link is created,
            % so that a new floe will be detected.
            if (l > 1) && (Links(1,l-1) == n-1),
                %   disp('Empty link')
                Links(:,l) = [n; 0];
                l = l+1;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            elseif (l == 1) && (Links(1,1) == 0),
                %   disp('Empty link for 1st cell')
                Links(:,1) = [n; 0];
                l = l+1;
                if l == length(Links(1,:)),
                    % Increasing size of Links
                    Links = horzcat(Links,zeros(2,lgth_1));
                end;
            end;
        end;
    end;

    % Creation or extension of the floes according to the Links matrix
    if i == 1, % Only new floes are created
        for n = 1:n_f,
            % Floe number, ice piece in line i-1, area; left side x-coord.
            F(:,n) = [n; n; (k_e(n)-k_s(n)+1)*dx_ice^2; (i-1)*dx_ice];
        end;
        f_e = n_f;  %Largest active floe number (last registered floe nbr)
        f_e_c = n_f;%Column index of the largest active floe number
    else
        F2 = F(2,:);
        for ll=1:(l-1),
            F_2p = F(2,:);
            if Links(2,ll) ~= 0,
                p = 1;
                while (F_2p(p) ~= Links(2,ll)) && (p < length(F_2p)),
```

```
                p = p+1;
            end;
            % if p >= length(F_2p),
            %     disp('Skipped ice piece')
            % end;

            % IF statement to check for ice pieces linked to several
            % floes
            if (ll==1) || ((ll>=2) && (Links(1,ll-1)~=Links(1,ll))),
                % IF statement to check for consecutive ice pieces
                % linked to the same floe
                if (ll==1)||((ll>=2) && (Links(2,ll-1)~=Links(2,ll))),
                    F2(p) = Links(1,ll);
                    F(3,p) = F(3,p)...
                      +(k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;
                else
                    % If two consecutive ice pieces for the same floe,
                    % no change and we keep the lowest number.
                    %disp('Consecutive ice pieces for the same floe')
                    lll = ll;
                    % Looking for the 1st ice piece of the series
                    while (lll > 1) && (Links(2,lll-1)==Links(2,lll)),
                        lll = lll - 1;
                    end;
                    F(3,p) = F(3,p)...
                      +(k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;
                    k_e(Links(1,lll)) = k_e(Links(1,ll));
                    k_e(Links(1,ll))=length(Vertic)+1; %Out of domain
                    k_s(Links(1,ll))=length(Vertic)+1; %Out of domain
                end;
            else
                % The current ice piece connects to two initially
                % independent floes
                % => the area of the two floes are combined into the
                % 1st one.
                % The residual floe number is set at zero and will
                % stay in F until the end of the calculation
                % disp(['Current ice piece connects to two ',...
                %       'initially independent floes'])
                F(1,p) = 0;

                pp = 1;
                while F_2p(pp) ~= F(2,p)-1,
                    pp = pp+1;
                end;

                F(3,pp) = F(3,pp) + F(3,p);
                F(:,p) = [];
                F2(p)  = [];
                f_e_c = f_e_c - 1;
            end;
        else %New floe
            %disp('New floe')
            f_e = f_e + 1;
            f_e_c = f_e_c + 1;
            %Floe nbr; ice piece in line i-1; area; left side x-coord.
            F(:,f_e_c) = [f_e;...
                Links(1,ll);...
                (k_e(Links(1,ll))-k_s(Links(1,ll))+1)*dx_ice^2;...
                (i-1)*dx_ice];
            F2(f_e_c) = Links(1,ll);
        end;
    end;
    F(2,:) = F2;
end;

% Check for floe breaking
x = (i-1)*dx_ice; % x-coordinate of the current line
q = 1;
  pp = zeros(1, f_e_c);
for p = 1:f_e_c,
  if ((x - F(4,p) + dx_ice) >= d_floe_target) && (F(1,p) ~= 0),
          %Floe breaks because of its size
          %disp(['Floe breaking (number ', num2str(F(1,p)), ')'])
    pp(q) = p;
    q     = q+1;
          k_s(F(2,p)) = 0;
          k_e(F(2,p)) = 0;
    elseif (F(3,p) == F_m1(3,p)) && (F(1,p) ~= 0),
          %Floe has not changed of area, meaning that it is isolated and
          %cannot grow anymore
          %disp(['Isolated floe (number ', num2str(F(1,p)), ')'])
    pp(q) = p;
    q     = q+1;
  end;
end;
  pp(q:end) = [];
  if f+length(pp) > length(G),
      % Increasing size of G
      G = horzcat(G,zeros(1,lgth_1));
  end;
G((f+1):(f+length(pp)))=F(3,pp);%We keep only the floe area in memory
F(:,pp) = [];
  F       = horzcat(F,zeros(4,length(pp)));%Keeping constant size for F
  f_e_c   = f_e_c - length(pp);
f       = f + length(pp);

% We remember the previous line for the calculation of next line
k_s_im1 = k_s;
k_e_im1 = k_e;
n_f_im1 = n_f;
  F_m1    = F;
end;

G((f+1):end) = [];

% OUTPUTS
disp(' - Floe size analysis :')
disp(['   + Calculated broken floes : ', num2str(f)])
disp(['   + Mean floe size: ', num2str(mean(G),5),' m^2.'])
disp(['   + Biggest : ', num2str(max(G)),' m^2.'])
disp(['   + Smallest : ', num2str(min(G)),' m^2.'])

% FIGURES
if fig_main,
    if not(fig_local),
        figure('Position', [0 0 600 450]),
        nbins = 30;

        subplot(10,1,1:7)
        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
            'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
            'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', {X_tick_mass})

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                          *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', {X_tick_momentum})

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')
    else
        figure('Position', [0 0 1000 450]),
        nbins = 30;

        subplot(10,6,[1:3 7:9 13:15 19:21 25:27 31:32 37:38])

        [n_G, x_G] = hist(G,nbins);
        bar(x_G,100*n_G./sum(n_G),1.0,'hist')
        grid on,
        xlabel('\textit{Floe area} $[m^2]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        ylabel('\textit{Occurence} $[\%]$', 'Fontsize', 12,...
                'interpreter', 'latex')
        title(['Resulting floe size - Reference : ', Ref],...
                'Fontsize', 12)

        set(gca,'xticklabel',num2str(get(gca,'xtick')'))
        ax_x_main      = gca;
        pos_ax_x_main = get(ax_x_main, 'Position');
        X_tick_main    = get(ax_x_main, 'XTick');
        X_lim_main     = get(ax_x_main, 'XLim');

        % Floe mass distribution
        ax_x_mass = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.12 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_mass = X_tick_main*(H_ice*rho_i+H_snow*rho_s)/1e3;
        set(ax_x_mass, 'XTickLabel', {X_tick_mass})

        xlabel('\textit{Floe mass} $[ton]$', 'Fontsize', 12,...
            'interpreter', 'latex')

        % Floe momentum distribution
        ax_x_momentum = axes('Position',...
        [pos_ax_x_main(1) pos_ax_x_main(2)-.24 pos_ax_x_main(3) 0.001],...
        'Color','none', 'XLim', X_lim_main);

        X_tick_momentum = X_tick_main*(H_ice*rho_i+H_snow*rho_s)...
                          *V_drift/1e3;
        set(ax_x_momentum, 'XTickLabel', {X_tick_momentum})

        xlabel('\textit{Floe momentum} $[ton\times m/s]$',...
                'Fontsize', 12,...
                'interpreter', 'latex')

        % Figure of the sampling area
        subplot(10,6,...
              [5:6 11:12 17:18 23:24 29:30 35:36 41:42 47:48 53:54 59:60])

        imagesc(DX, DY, ICE)
        set(gca,'YDir','normal')
        colormap(gray)
        daspect([1 1 1])
        title('Sampling area')
        xlabel('\textit{x} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
        ylabel('\textit{y} $[m]$', 'Fontsize', 12, 'interpreter', 'latex')
    end;
end;
end
```

# E EXECUTABLE FOR ESTIMATION OF RUDDER FORCES

## Example of input files

### E.1 rpsim2.txt

```
# Please type missing information.
# Positive sense of coordinates: ahead, to port, upwards.
# Origin of global reference frame in aft perpendicular, in base.
#
PropellerName = prop
PropellersRudder = rud
#
# Propeller ahead of aft perpendicular:
PropellerPositionX = 0.
#
PropellerPositionY = 0.
#
# Propeller above base:
PropellerPositionZ = 1.5
#
# Euler angles in case of inclined propeller:
PropellerPositionPhi = 0.
PropellerPositionTheta = 0.
PropellerPositionPsi = 0.
#
#
PropellerWakeFraction = 0.09
#
# Propeller data:
PropellerDiam = 1.500
PropellerNBlades = 4.
PropellerPD = 0.95
PropellerAeA0 = 0.5
PropellerHubDRatio = 0.2
PropellerSense = r
PropellerDucted = no
PropellerCPP = no
PropellerMaxRPM = 200
#
# optionally: open water diagram of propeller as table:
PropellerFile = prop2.txt
#
# thrust deduction fraction ahead and astern:
PropellerTdfAhead = 0.02
PropellerTdfAstern = 0.25
#
RudderName = rud
RuddersPropeller = prop
#
# origin of rudder's local reference frame ahead of aft
      perpendicular:
RudderPositionX = -1.
#
RudderPositionY = 0.
#
# origin of rudder's local reference frame above base:
RudderPositionZ = 1.5
#
# Euler angles in case of inclined rudder:
RudderPositionPhi = 0.
RudderPositionTheta = 0.
RudderPositionPsi = 0.
#
# rudder contour in its local reference frame:
RudderXVertex = 0.3
RudderZVertex = -0.6875

RudderXVertex = 0.3
RudderZVertex = 0.6875

RudderXVertex = -1.
RudderZVertex = 0.6875

RudderXVertex = -1.3
RudderZVertex = 0.523

RudderXVertex = -1.3
RudderZVertex = -0.523

RudderXVertex = -0.54
RudderZVertex = -0.6875

RudderStallAngle = 30.
#
# optionally: rudder characteristics as table:
RudderFile = rudder2.txt
```

### E.2 prop2.txt

```
kT at different pitches [deg]
J     0.
0       0.45297
0.05    0.42873
0.1     0.40441
0.15    0.37996
0.2     0.35533
0.25    0.33051
0.3     0.30552
0.35    0.28037
0.4     0.25508
0.45    0.22967
0.5     0.20417
0.55    0.17862
0.6     0.15305
0.65    0.12747
0.7     0.10186
0.75    0.0762
0.8     0.05048
0.85    0.0247
0.9    -0.00113

10*kQ at different pitches [deg]
J     0.
0       0.62057
0.05    0.59052
0.1     0.56046
0.15    0.53036
0.2     0.50022
0.25    0.47003
0.3     0.43977
0.35    0.40944
0.4     0.37902
0.45    0.34851
0.5     0.31789
0.55    0.28717
0.6     0.25635
0.65    0.22543
0.7     0.19441
0.75    0.16329
0.8     0.13207
0.85    0.10077
0.9     0.06941
```

## E.3 rudder2.txt

```
alpha    cD        cL       cQN
[deg]   [-]       [-]       [-]
-180.0   0.0406   0.120  0
-177.5   0.0450   0.365  0.006
-175.0   0.0639   0.550  0.008
-172.5   0.1051   0.751  0.009
-170.0   0.1631   0.776  0.01
-167.5   0.2210   0.723  0.01
-165.0   0.2783   0.673  0.01
-162.5   0.3214   0.636  0.01
-160.0   0.3538   0.630  0.01
-155.0   0.4445   0.695  0.01
-150.0   0.6220   0.770  0.01
-145.0   0.8175   0.813  0.01
-140.0   0.9406   0.832  0.01
-135.0   1.1152   0.849  0.01
-130.0   1.2439   0.786  0.01
-125.0   1.3666   0.709  0.01
-120.0   1.5314   0.640  0.01
-115.0   1.6097   0.527  0.01
-110.0   1.6718   0.410  0.01
-105.0   1.7134   0.273  0.01
-100.0   1.7356   0.151  0.01
-95.0  1.7426   0.015  0.008
-90.0  1.8133   -0.117   0
-85.0  1.7644   -0.251   -0.008
-80.0  1.7146   -0.385   -0.01
-75.0  1.6478   -0.519   -0.01
-70.0  1.5587   -0.617   -0.01
-65.0  1.4904   -0.738   -0.01
-60.0  1.3535   -0.838   -0.01
-55.0  1.2904   -0.905   -0.01
-50.0  1.1588   -0.987   -0.01
-45.0  0.9893   -0.966   -0.01
-40.0  0.8345   -0.918   -0.01
-35.0  0.6679   -0.846   -0.01
-30.0  0.4952   -0.752   -0.01
-27.5  0.4280   -0.730   -0.01
-25.0  0.3646   -0.718   -0.01
-22.5  0.3136   -0.734   -0.01
-20.0  0.2699   -0.787   -0.01
-17.5  0.1684   -0.996   -0.01
-15.0  0.0795   -1.204   -0.01
-12.5  0.0487   -1.109   -0.01
-10.0  0.0346   -0.946   -0.01
-7.5   0.0240   -0.725   -0.01
-5.0   0.0134   -0.475   -0.008
-2.5   0.0125   -0.237   -0.006
0.0  0.0115   0.0  0
2.5  0.0125   0.237  0.006
5.0  0.0134   0.475  0.008
7.5  0.0240   0.725  0.01
10.0   0.0346   0.946  0.01
12.5   0.0487   1.109  0.01
15.0   0.0795   1.204  0.01
17.5   0.1684   0.996  0.01
20.0   0.2699   0.787  0.01
22.5   0.3136   0.734  0.01
25.0   0.3646   0.718  0.01
27.5   0.4280   0.730  0.01
30.0   0.4952   0.752  0.01
35.0   0.6679   0.846  0.01
40.0   0.8345   0.918  0.01
45.0   0.9893   0.966  0.01
50.0   1.1588   0.987  0.01
55.0   1.2904   0.905  0.01
60.0   1.3535   0.838  0.01
65.0   1.4904   0.738  0.01
70.0   1.5587   0.617  0.01
75.0   1.6478   0.519  0.01
80.0   1.7146   0.385  0.01
85.0   1.7644   0.251  0.008
90.0   1.8133   0.117  0
95.0   1.7426   -0.015   -0.008
100.0  1.7356   -0.151   -0.01
105.0  1.7134   -0.273   -0.01
110.0  1.6718   -0.410   -0.01
115.0  1.6097   -0.527   -0.01
120.0  1.5314   -0.640   -0.01
125.0  1.3666   -0.709   -0.01
130.0  1.2439   -0.786   -0.01
135.0  1.1152   -0.849   -0.01
140.0  0.9406   -0.832   -0.01
145.0  0.8175   -0.813   -0.01
150.0  0.6220   -0.770   -0.01
155.0  0.4445   -0.695   -0.01
160.0  0.3538   -0.630   -0.01
162.5  0.3214   -0.636   -0.01
165.0  0.2783   -0.673   -0.01
167.5  0.2210   -0.723   -0.01
170.0  0.1631   -0.776   -0.01
172.5  0.1051   -0.751   -0.009
175.0  0.0639   -0.550   -0.008
177.5  0.0450   -0.365   -0.006
180.0  0.0406   -0.120   0
```