

Rotare

---

Documentation

Thomas Lambert

*February 13, 2023*

Version: v1.0.0 -**DEV**

© Copyright 2022-2023 University of Liège

The entire code for the software ROTARE is provided under the MIT license.

This documentation and all the related material (figures, etc.) are provided under the Creative Commons Attribution-ShareAlike 4.0 license (CC BY-SA 4.0).



# Contents

<b>I</b>	<b>User manual</b>	<b>4</b>
<b>1</b>	<b>Install guide</b>	<b>5</b>
1.1	Manual install (recommended) . . . . .	5
1.2	Install using git . . . . .	5
1.3	Update . . . . .	6
1.3.1	Before updating . . . . .	6
1.3.2	Manual update . . . . .	6
1.3.3	Update using git . . . . .	7
<b>2</b>	<b>Usage</b>	<b>8</b>
<b>3</b>	<b>Configuration file</b>	<b>9</b>
3.1	Configuration validation . . . . .	9
3.2	Input variables . . . . .	9
3.2.1	General simulation options (Sim) . . . . .	10
3.2.2	Models and solver options (Mod) . . . . .	12
3.2.3	Flow . . . . .	13
3.2.4	Operating points . . . . .	13
3.2.5	Airfoil . . . . .	14
3.2.6	Blade . . . . .	16
<b>4</b>	<b>Troubleshooting</b>	<b>18</b>
4.1	<Library> cannot be found or cannot be imported . . . . .	18
4.2	Is <feature> going to be implemented? When? . . . . .	18
<b>II</b>	<b>Technical documentation</b>	<b>19</b>
<b>5</b>	<b>Solvers</b>	<b>21</b>
5.1	Coordinates systems . . . . .	21
5.2	Rotor coefficients . . . . .	21
5.3	Blade Element theory . . . . .	22

5.4	Momentum theory . . . . .	24
5.5	Solvers . . . . .	25
5.5.1	Leishman solver . . . . .	26
5.5.2	Induction factor . . . . .	27
5.5.3	Induced velocities . . . . .	28
5.5.4	Stahlhut solver . . . . .	29
5.6	Coaxial rotors . . . . .	29
5.7	Oblique flows . . . . .	29
<b>6</b>	<b>Extensions and corrections</b>	<b>30</b>
6.1	Tip/hub loss . . . . .	30

# Nomenclature

## Latin Letters

$\mathcal{J}$	Advance ratio	–
$A$	Area	$\text{m}^2$
$a$	Axial induction factor	–
$c_d$	Drag coefficient	–
$c$	Rotor chord	$\text{m}$
$c_l$	Lift coefficient	–
$c_{l,\alpha}$	Lift curve slope	–
$c_m$	Moment coefficient	–
$C_P$	Power coefficient	–
$C_T$	Thrust coefficient	–
$C_Q$	Torque coefficient	–
$\rho$	Density	$\text{kg}/\text{m}^3$
$D$	Rotor diameter	$\text{m}$
$\mathcal{D}$	Drag	$\text{N}$
$\mathcal{L}$	Lift	$\text{N}$
$\mathcal{M}$	Mach number	–
$\dot{m}$	Mass flow rate	$\text{kg s}^{-1}$
$N_b$	Number of blades	–

$\mathcal{P}$	Power	W
$F$	Prandtl's tip-loss factor	–
$y$	Absolute radial position	m
$R$	Rotor radius	m
$r$	Nondimensional radial position, $\frac{y}{R}$	–
$Re$	Reynolds number	–
$n$	Angular velocity	$s^{-1}$
$b$	Tangential induction factor	–
$\mathcal{T}$	Thrust	N
$\mathcal{Q}$	Torque	N m
$V_\infty$	Freestream velocity	$m s^{-1}$
$V$	Axial velocity	$m s^{-1}$
$v_i$	Induced axial velocity	$m s^{-1}$
$u_i$	Induced tangential velocity	$m s^{-1}$
$W$	Relative velocity	$m s^{-1}$
$v_w$	Axial velocity in the far wake	$m s^{-1}$
$u_w$	Tangential velocity in the far wake	$m s^{-1}$
$U$	Tangential velocity	$m s^{-1}$

### **Greek Letters**

$\phi$	Induced angle	
$\Omega$	Angular velocity	$s^{-1}$
$\alpha$	Angle of attack	

$\zeta$	Rotor angle of attack	
$\psi$	Azimuthal angle	
$\beta_0$	Collective pitch angle	
$\lambda$	Inflow ratio, $\frac{V}{\Omega R}$	$s^{-1}$
$\lambda$	Induced inflow ratio, $\frac{v_i}{\Omega R}$	$s^{-1}$
$\beta$	Pitch angle	
$\sigma$	Rotor solidity, $\frac{N_b c}{\pi R}$	—
$\sigma_l$	Local solidity, $\frac{N_b c(y)}{\pi R}$	—
$\Lambda$	Blade sweep angle	
$\xi$	Swirl ratio, $\frac{U}{\Omega R}$	$s^{-1}$
$\xi$	Induced swirl ratio, $\frac{u_i}{\Omega R}$	$s^{-1}$
$\chi$	Twist angle (stagger for propellers)	

### Subscripts

- $( )_d$  Far downstream of the rotor (slipstream)
- $( )_u$  Upstream of the rotor

# Introduction

ROTARE is a feature-rich and open-source implementation of the Blade Element Momentum Theory (BEMT) in MATLAB.

This software can be used for the analysis and the design of all kinds of rotors: helicopters main/tail rotors, aircraft propellers, wind/tidal turbines, etc.

ROTARE was developed primarily for teaching purposes at the University of Liege (Belgium) by Thomas Lambert during his Ph.D. The code was later extended to add different solvers, many extensions to the base methodology and to support more complex geometries. It is now a complete analysis tool that can be used in a wide range of applications outside of academia.

The present documentation is divided in two main parts:

1. **The user manual** (Part I), with practical details about the installation and usage of the code.
2. **The technical manual** (Part II), with details regarding the code architecture and all the theory underlying the implementation.

## Disclaimer

ROTARE is still currently under heavy development. The first truly stable version will be numbered **1.0.0**. Versions prior to this one are **not validated and may contain errors and a few bugs**.

Starting from v1.0.0, this documentation will reflect the state of the software at each release. Development version of the documentation will be kept as updated as possible with the code. These will be clearly labeled **-DEV** in the titlepage.



## Features

ROTARE currently supports blade geometries with varying twist, chord and airfoil. At the moment, the software is only able to model single isolated rotors. These can be studied in steady conditions (*e.g.*, hovering helicopter) or axial flows (*e.g.*, aircraft propeller, helicopter in climb). More complex cases (oblique flows, coaxial rotors, etc.) will be added later on.

The software currently supports a correction for the tip and hub losses. A few other corrections and extensions will be implemented shortly, such as: tip/hub losses, compressibility effects, spinner effects, etc (see Chapter 6 for more details on these extensions).

Different solvers for the BEMT equations are implemented in the software (see Chapter 5 for complete description). While they all solve the same initial set of equations, they differ on the methodology for the resolution or the hypotheses made to solve the equations. Even though some solvers are clearly superior to others, this redundant implementation is especially useful for teaching purposes. Indeed, it allows to compare the quality of the results, the convergence or the effect of additional hypotheses.

While ROTARE is not specifically meant for wind turbine, it can model their behavior as long as the user specifies all operating conditions (the same way we would for an helicopter rotor). ROTARE is currently not able to model "windmilling" or "autorotation". This will only be added at a later stage of the development.

# Part I

---

User manual

# Install guide

ROTARE does not require any specific external program to be installed. The install process is just a question of downloading the MATLAB script and functions and placing them in at the correct place.

There are two possibilities to install ROTARE on your system.

1. Downloading the source code of the latest stable release from the release page
2. Cloning the repository with git

The manual download of files is recommended as it is easier, will not require any specific knowledge about git and will always download the latest *stable* release of ROTARE. The git method should be reserved to more advanced users. With this you will have the latest developments, but some bugs are more likely to happen.

## 1.1 Manual install (recommended)

For the manual install you must download the latest stable release of ROTARE and extract the archive on your computer.

⚠ In order to download the *complete* software (*i.e.*, the code and the libraries) you must download the item called **Complete Code: rotare-x.x.x.zip**. The items marked as *source code* under *assets* do not contain the mandatory libraries and will prevent ROTARE from running properly.

## 1.2 Install using git

Navigate to the directory where you want to place ROTARE and then simply use `git clone --recursive` to download the source code and its libraries.

```
1 cd ~/Documents/  
2 git clone --recursive https://gitlab.uliege.be/rotare/rotare
```

This is it. You now have the most up-to-date version of ROTARE installed on your computer. Note that this version may not be completely stable.

You can open Matlab and navigate to `Documents/rotare/src/` to start using ROTARE.

## 1.3 Update

In principle, the update process should never touch any of your files. However, before any update, it is advised to **make a backup of your existing configuration files and results** outside of ROTARE's working directory.

### 1.3.1 Before updating

Prior to any update, it is heavily recommended to review the changelog (and/or the release notes) and make sure the update does not introduce some breaking changes. Normally, this should only be the case for Major versions of the code (e.g., going from 1.x.x to 2.x.x).

If any breaking change is documented, please review the changelog or this documentation and proceed to the necessary adaptations.

### 1.3.2 Manual update

Repeat the steps described in Section 1.1 and copy the new files over the older ones. You could also just extract it somewhere else and move your configurations files and results to the new directory.

Note that if you modified some files of the source code, a simple copy/paste will erase these changes.

### 1.3.3 Update using git

Navigate to ROTARE's directory and simply run

```
1 | git pull --recurse-submodules
```

This will update ROTARE and its libraries. This should not modify your configuration files or the results you generated. However, if you modified some portions of the source code, it may fail to update. In that case, follow git's instructions to either remove your changes, or commit them and resolve the merge conflict.

## Usage

ROTARE is entirely controlled by a single input file, where the user specifies the simulation parameters, the rotor geometry, the flow conditions, etc. See Chapter 3 for details about this configuration file. This is the only place that requires user's attention.

In order to use ROTARE, make sure you are in the `rotare/src/` directory, or that it is in MATLAB's Path.

ROTARE can be called directly with a configuration file or without one. If no configuration file is provided, the user will be prompted to select a configuration file manually.

```
1 % Manual selection of input file
2 rotare
3 % With a specific config file
4 rotare('configs/my_config.m')
```

ROTARE comes with a few test configuration files (a template file and some reproduction of experimental results). You can verify that everything is working as intended by running any of these files:

```
1 rotare('configs/template.m')
2 rotare('configs/caradonna1981.m')
3 rotare('configs/knight1937.m')
```

# Configuration file

A simulation with ROTARE is controlled by a single configuration file. This file contains all data related to the behavior of the software, the models and solvers to use, the flow, the operating points to analyze and the rotor geometry itself. These five types of input are collected inside five different structures for easier handling in the code.

This input file should be a MATLAB script (not a function). Examples of configuration are given in `src/configs/`. In order to always have a fully-defined and working configuration available, it is suggested to just copy the template and edit your copy instead of directly working in the template (ROTARE will nag you if you are using the template anyway).

## 3.1 Configuration validation

Before initiating a simulation, the user input file is passed through a validation function automatically. This function checks if all inputs are properly defined (no missing parameters) and formatted (correct type, dimensions, etc) before starting the simulation. It also fixes small typos in the string parameters whenever possible to ensure proper behavior of the software.

If the validation of a parameter fails, a comprehensive error message will be returned. This should help the user fix their issues easily.

The validation function will also return preliminary warnings whenever some parameters may lead to potential issues (*e.g.*, not enough elements to properly define the geometry, etc.).

## 3.2 Input variables

In this documentation, all input parameters are structured as in the following example.

**Option Name** [unit] ..... Example or allowed values  
 A brief description of the option with further indications regarding the input values if necessary.

### 3.2.1 General simulation options (Sim)

The structure `Sim` contains all general parameters to tailor the behavior of the software. It is itself made of sub-structures, ordered by themes.

#### **Sim.Save**

This structure holds all parameters related to the automated saving of the simulation results.

**Sim.Save.autosave** ..... `true,false`  
 Automatically save the simulation results in a MAT-file. This MAT-file can be re-imported later for post-processing and result analysis.

**Sim.Save.override** ..... `true,false`  
 Overwrite existing results when saving automatically. Results are only overwritten if a file with matching name exist.  
 If set to `false`, a number will be added at the end of the filename and incremented upon each save.

**Sim.Save.dir** ..... `'../results/'`  
 Directory where the saved results will be exported.

**Sim.Save.filename** ..... `'myfile'`  
 Filename to use for the saved results.

**Sim.Save.appendInfo** ..... `true,false`  
 Automatically append some simulation information to the `FilenameBase`.

**Sim.Save.prependTime** ..... `true,false`  
 Automatically prepend a time code (format `YYYYMMDDHHmmSS` at the beginning of the filename.

**Sim.Save.timeFormat** ..... `YYYYMMDDHHmmSS`  
 Format of the timecode to use when `Sim.Save.prependTime=true`.



## Sim.Out

This structure holds all parameters related to the display of the simulation's results.

**Sim.Out.showPlots** ..... true,false  
Display all plots and graphs at the end of the simulation.

**Sim.Out.show3D** ..... true,false  
Show a 3D view of the rotor and one with a single isolated blade.

**Sim.Out.hubType** ..... See below  
Type of hub (cone) to display on the 3D graph. See Nose cone design (wikipedia) for representation of the nose cones.

Allowed values: 'none', 'cylinder', 'conic', 'blunted\_conic', 'bi-conic',  
'tangent\_ogive', 'blunted\_tangent\_ogive', 'secant\_ogive\_regular', 'secant\_ogive\_bulge',  
'elliptical', 'parabolic', 'power\_series', 'lv-haack', 'vonkarman'.

*The nose cones are only used for a more realistic visual representation of the rotor. The type of hub itself has absolutely no impact in the actual computation of the BEMT results.*

**Sim.Out.console** ..... true,false  
Print the simulation results in the console.

**Sim.Out.verbosity** ..... min,all  
Verbosity level for the console output.

## Sim.Warn

Enable or disable specific warnings during the simulations.

Note that only *low severity* warnings can be disabled this way from the configuration file. These warnings usually indicate a poor rotor design or a poor operating condition for a proper rotor. As these conditions are to be expected when the full operating map of the rotor is being simulated, these low severity warnings can be silenced to avoid cluttering the console.

However, the code may also output more severe warnings. These are typically related to violated hypotheses and indicate unreliable results. As they are considered more serious, it is by purpose that they can not be silenced easily through the configuration file. Even though it is *heavily discouraged*, it is still possible to use MATLAB built-in functions to disable these critical warnings.

**Sim.Warn.sonicTip** ..... true,false  
Warns the user if the blade tip is trans/super sonic.

### Sim.Misc

Other miscellaneous parameters.

**Sim.Misc.nonDim** ..... 'US', 'EU'  
The non dimensionalization factor used to get the thrust, torque and power coefficient is not always the same between the US and the rest of the world. While they differ only from a factor 0.5, this can lead to difficulties when comparing with existing experimental data.

*See section 5.2 for details about the rotor coefficients.*

**Sim.Misc.appli** ..... 'prop', 'heli', 'turbine'  
Specifies the type of application for the rotor studied. This impacts the definition of the forces, moments and power coefficients calculated. It is also used to display the 3D view in the appropriate position and to ensure proper sign for the output variables, etc.

*See section 5.2 for details about the rotor coefficients.*

## 3.2.2 Models and solver options (Mod)

The structure `Mod` contains the parameters for the solvers and the extensions/corrections to apply. It also specifies the numerical limits (number of iterations, precision, etc.).

**Mod.solvers** ..... 'leishman', 'prop', 'turbine', 'stahlhut', 'all'  
Type of solver to use. The solvers are described in length in Chapter 5. *It is possible to provide a cell array with multiple solvers. Rotare will then loop for all solvers.*

### Mod.Ext

Parameters related to the extensions and corrections to apply to the base BEMT. See section 6 for details regarding these models.

**Mod.Ext.losses** ..... 'none', 'hub', 'tip', 'both'  
Type of losses to consider (using Prandtl formula).

## Mod.Num

Numerical limits for the simulations.

**Mod.Num.convCrit** [-] ..... 1e-4

Value for the convergence criterion to use when doing iterative processes. Note that this criterion will be used to assess the *relative* error between two iterations. A precision of 0.01% is often low enough to get valid results.

**Mod.Num.maxIter** [-] ..... 1e3

Maximum iterations allowed when doing iterative processes. If this number is reached, the code will output an error and stop its execution.

**Mod.Num.relax** [-] ..... 0.1

Relaxation factor to use in order to ease the convergence of iterative processes. A low number would increase chance of success of the convergence process at the cost of a slower converge. A higher number (up to 1) will proceed faster, but the solution may diverge.

### 3.2.3 Flow

The structure `Flow` contains the parameters related to the flow itself.

**Flow.fluid** ..... 'air', 'seawater', 'freshwater'

Nature of the fluid. This is used to determine the density and viscosity of the fluid. Note that if the fluid is air, the altitude (see `Op.alt`) is also used in order to determine the proper density and viscosity (using ISA tables).

### 3.2.4 Operating points

The structure `Op` determines the various operating points of the rotor(s). These four variables can be specified as vectors in order to study a given rotor geometry over multiple operation points (thus creating a whole operating map of the rotor). Obviously, it is also possible to simply specify scalars. In that case, ROTARE will analyze of the rotor at one single point.

Note that ROTARE will loop over *every combination* of these four operating points. Therefore, the total number of simulations can become very large should you decide to study lots of these points.

**Op.speed** [m/s] ..... Vector ( $1 \times N_2$ )

Axial velocity of the fluid.

**Op.altitude** [m] ..... Vector ( $1 \times N$ )

The rotor altitude (e.g., flight altitude, wind turbine elevation).

This is only used to get a better estimation of the air density. If `Flow.fluid` is not air, this is not used.

**Op.rpm** [rpm] ..... Vector ( $1 \times N$ )

The rotor angular velocity in RPM.

**Op.collective** [deg] ..... Vector ( $1 \times N$ )

The collective pitch setting for the rotor. If the rotor has no collective pitch setting (e.g., drone propeller), it is advised to let this option at 0 and only specify the twist of the rotor in the `Blade` structure.

### 3.2.5 Airfoil

Data regarding the various airfoils that will be used along the blade. Note that multiple airfoils can be specified. In that case, just use multi-dimension structures (i.e., `Airfoil(1)`, `Airfoil(2)`, ...).

**Airfoil.coordFile** ..... 'airfoil\_data/naca0012.dat'

Name of the file with the airfoil coordinates points. At the moment, this file is only used to draw the 3D view of the rotor, but it is still a mandatory input. The best source for such data files is to directly get the UIUC Airfoil Coordinates Database. The data can be formatted either following Selig or Ledneicer convention (the two types of format found on UIUC Database).

**Airfoil.polarType** ..... 'file', 'polynomial'

Type of polar input ROTARE should be using.


- **file**: A structure containing the polars should be passed to ROTARE.
- **polynomial**: The  $C_l$  and  $C_d$  are given as polynomial expressions of  $\alpha$ .

**Airfoil.polarFile** ..... 'airfoil\_data/naca0012-polar.mat'

Name of the files with the airfoil polars. This file should contain a Matlab structure called `Polar`. Such file can be obtained by generating the airfoil polars with XFOIL or XFLR5 and then exporting them with the `xf2mat` utility that can be found in the

free and open-source `matlab_airfoil_toolbox`. Note that this toolbox is a required library for ROTARE and should already be present and usable in your installation.

**Airfoil.extrapMethod** ..... See below


 Only used when `Airfoil.polarType='file'`.

Extrapolation method to use in order to recover the lift and drag coefficients from incomplete polars. Possible choices are:

- **none**: no extrapolation allowed. Will return an error if the solution requires the calculation of an angle of attack outside of the range provided by the user.
- **spline**: spline extrapolation if angle of attack is outside the range of angles given in the input polar. Strongly discouraged.
- **viterna**: Extrapolation of the polars over the entire range of angles of attack according to Viterna formulas ???. Although this is not perfectly correct (as it is the case for any extrapolation), this option is the one that is closer to the reality. This is particularly useful when doing a sweep of operating conditions and many off-design points must be studied.

Note that convergence may be more trickier to achieve in some edge cases and off-design analysis with the low-order extrapolation methods. In operation at the design point, extrapolation should not be required at all (as long as the polars are provided for values between the minimum lift and the stall point).

**Airfoil.clPoly** [ $\text{deg}^{-1}$ ] ..... Vector

 Only used when `Airfoil.polarType='polynomial'`.


Coefficients for the polynomial form of the lift coefficient,  $C_l$ .

The vector should be in the form  $[p_1, p_2, \dots, p_n]$  to represent the polynomial

$$C_l = \sum_{i=1}^n p_i \alpha^{n-i}$$

, where  $\alpha$  is the angle of attack in degree.

**Airfoil.cdPoly** [ $\text{deg}^{-1}$ ] ..... Vector

 Only used when `Airfoil.polarType='polynomial'`.

Same as `Airfoil.clPoly`, but for the drag coefficient.

### 3.2.6 Blade

The structure `Blade` contains the parameters related to the rotor and blade geometry. If there are multiple rotors (*i.e.*, currently only for coaxial case), you must specify these parameters for both rotors. If the two rotors are the same, a simple way to do that consists in adding `Blade(2) = Blade` after the definition of the first rotor.

**⚠ The blade dimensions are specified through a vector of at least two elements.** The first element always correspond to the blade root, the last one is always the blade tip. The blade geometry will then be constructed by interpolating points at the different segments, using a spline interpolation (it is therefore not necessary to add every single blade section in these vectors). The vectors can contain as many points as desired in order to refine the mesh or better control the overall geometry of the blade.

Note that the vectors must all have the same dimensions as `Blade.radius`. The other variables (`chord`, `twist`, `iAf`) all correspond to the value in the base stations defined in `Blade.radius`.

**Blade.nBlades** [-] ..... 3  
Number of blades on the rotor.

**Blade.pitchRef** ..... 'zerolift', 'chordline'  
It is common to define the pitch angle of a blade element with respect to the zero-lift angle of its airfoil instead of the chord line. This parameter ensures the correct reference is taken.

**Blade.radius** [m] ..... [ $r_{\text{root}}, \dots, r_{\text{tip}}$ ]  
Radial position of the elements. The first element corresponds to the root of the blade (include the cutout) and the last one corresponds to the blade tip. Intermediary points may be added to tweak more precisely the blade geometry.

**Blade.chord** [m] ..... [ $c_{\text{root}}, \dots, c_{\text{tip}}$ ]  
Chord of the elements.

**Blade.twist** [deg] ..... [ $\theta_{\text{root}}, \dots, \theta_{\text{tip}}$ ]  
Twist (or stagger) angle of the elements. Note that if `Mod.collective` is not 0, the actual pitch of the elements will be the sum of the twist and the collective.

**Blade.iAirfoil** ..... [ $i_{\text{root}}, \dots, i_{\text{tip}}$ ]  
Index of the `Airfoil(i)` to use for each element. As there is no simple way to interpolate between different airfoils, the same airfoil will be applied on all sections until a new airfoil is specified.

Example:

Section	1	2	3
Radius	0.1	0.5	1
Airfoil	1	2	3

In that case, the `Airfoil(1)` will be applied for all elements between  $[0.1; 0.5[$  m, the `Airfoil(2)` will be used for all elements between  $[0.5; 1[$  m and the `Airfoil(3)` will be used for the tip.

**Blade.nElem** [-] ..... 100  
Number of blade elements to use in the Blade Element Method. The elements will be linearly spaced along the span of the blade (defined by the two bounds of `Blade.radius`).

**Blade.hubPos** [m] ..... [0, 0, 0]  
Coordinates of the rotor center point. This parameter is important for multi-rotor systems. For single rotors, it is discarded.

# Troubleshooting

This chapter lists the most common issues encountered when using ROTARE. If you find an issue that is not documented here, please check the issue tracker and fill a new issue report if applicable. You can also contact me directly at t.lambert@uliege.be.

The software contains many checks and is able to detect a common issues by itself. Please make sure you read the error message properly as they often indicate possible solutions.

## 4.1 <Library> cannot be found or cannot be imported

ROTARE depends on some libraries to conduct some simple tasks (mostly airfoil and polars manipulations). It seems that these were not installed properly. Please refer to Chapter 1 for the correct install procedure.

If ROTARE was cloned using git but without the `--recursive` tag, you can simply call the following to download the libraries as well:

```
1 | git submodule update --init
```

If ROTARE was installed manually, you probably did not download the proper archive. Please download the archive called **COMPLETE CODE: rotare-X.X.X.zip** and not the "Source code" under "Assets".

## 4.2 Is <feature> going to be implemented? When?

A basic roadmap can be found in the root of the repository. This outlines the planned developments and classes them in the most likely order of release. The dates of the releases are voluntarily left out to prevent breaking any promises.



# Part II

---

Technical documentation

# Introduction

This second part of ROTARE's documentation concerns the technical aspect of the code and its implementation. The goal is not to give a full lecture on the Blade Element Momentum Theory, but rather to write out the main equations and detail the way they are implemented and solved within ROTARE.

The first chapter details the architecture of the code itself and is more aimed at developers that want to extend the possibilities offered by the software.

The following the chapters are directed towards users that want an in-depth knowledge of the equations and solving process implemented in ROTARE. Finally, some basic validation cases are provided against well-known literature examples.

## Disclaimer

This section will be written at a later stage in the development. It currently contains small errors, notes, etc and has not be proofread.

# Solvers

FiXme: While ROTARE does not fully support coaxial rotors or oblique flows yet, the current documentation is written as if it was already the case.

FiXme!

## 5.1 Coordinates systems

The rotors may not always be aligned with the freestream. It is therefore important to properly define the reference frames before developing the equations. In specific, the first reference frame will simply be the internal one. The second frame is defined as the tip plane path of the rotor and will be denoted TPP. This frame is defined solely by the rotor disk. In oblique flows, it is angled with respect to the inertial frame, while in axial flows it lies perpendicular to the freestream. [FiXme: Add schematics](#)

FiXme!

## 5.2 Rotor coefficients

In order to better compare different rotors or their performance under different conditions, it is often best to express the forces by means of non-dimensional coefficients. While the principle is the same for all applications, different factors are used to non-dimensionalize these forces in rotors, propellers or turbine applications. Moreover, the general convention applied in the United States for rotor differs from the one used in the rest of the world [Lei06], where an additional one-half factor is used in the denominator.

The Table 5.1 lists the main coefficients in the US conventional notation. The other forces (longitudinal, lateral) or moments (rolling, pitching) are defined similarly to the thrust and torque coefficients respectively.

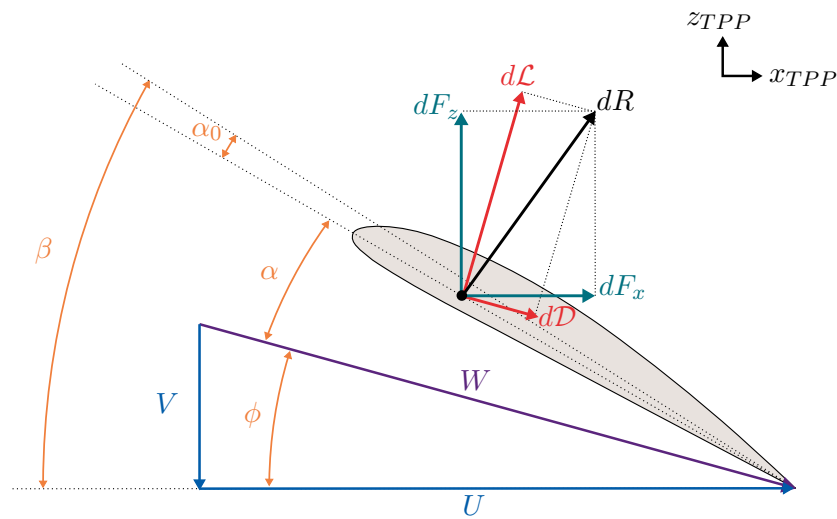
Note that the angular velocity in the rotor notation is  $\Omega$ , expressed in [1/sec] while for the propellers it is  $n$ , expressed in [RPS].

**Table 5.1.:** Coefficients definitions for rotors, propellers and turbines – US notation.

Measure	Rotors	Propellers	Turbines
Thrust coeff, $C_T$	$\frac{\mathcal{T}}{\rho A (\Omega R)^2}$	$\frac{\mathcal{T}}{\rho n^2 D^4}$	FiXme: todo
Thrust torque, $C_Q$	$\frac{\mathcal{Q}}{\rho A \Omega^2 R^3}$	$\frac{\mathcal{Q}}{\rho n^2 D^5}$	FiXme: todo
Thrust power, $C_P$	$\frac{\mathcal{P}}{\rho A (\Omega R)^3}$	$\frac{\mathcal{P}}{\rho n^3 D^5}$	FiXme: todo

### 5.3 Blade Element theory

The Blade Element Theory postulates that a rotor can be represented as a collection of 2D elements which are radially distributed over the blade. It further assumes that the performance of an individual element is completely independent of the influence of the other elements. Each element can therefore be represented as a simple 2D airfoil section.



**Figure 5.1.:** Velocity diagram for a blade element  $dy$  at location  $y$ .

The representation in Figure 5.1 showcases a single blade element with its associated velocity triangles and forces. The element lies at a total pitch angle  $\beta$  with respect to the rotor disk. This pitch can be decomposed in different contributions:

$$\beta = \chi + \beta_0 \tag{5.1}$$

Where  $\chi$  is the twist angle of the blade (*i.e.*, a radial variation of pitch along the blade) and  $\beta_0$  is the collective pitch of the blade (*i.e.*, a modification of the pitch constant over the entire blade). Note that, by convention, the pitch angle is usually given with respect to the *zero-lift line* and not the *chord line* of the airfoil.

The resultant velocity is the sum of the velocity components acting in the  $x_{TTP}$  and  $z_{TTP}$  directions, respectively  $U$  and  $V$ .<sup>1</sup>

In oblique flows (*i.e.*, when the flow is not perfectly perpendicular to the rotor disk), the tangential velocity is dependent on blade instantaneous position, which can be represented through the azimuthal angle  $\psi$ .

$$V = V_\infty \sin \zeta + v_i \quad (5.2)$$

$$U = V_\infty \cos \zeta \sin \psi + \Omega y - u_i \quad (5.3)$$

$$W = \sqrt{V^2 + U^2} \quad (5.4)$$

The inflow angle (sometimes called *induced angle*)  $\phi$  expresses the angle between the resultant velocity and the rotor disk:

$$\tan^{-1} \phi = \frac{V}{U} \quad (5.5)$$

By convention,  $\phi$  is positive when  $W$  is directed towards the disk and negative when  $W$  points outwards.

The effective angle of attack at the element can therefore be obtained by the difference of the element's pitch and the inflow angle. Note that, as the pitch angle is often given with respect to the airfoil *zero-lift line*, the zero-lift angle must be subtracted as well.

$$\alpha = (\beta - \alpha_0) - \phi \quad (5.6)$$

The angle of attack, along with the Reynolds number can be used to determine the value of the airfoil lift and drag coefficient. The airfoil polar coefficients are usually given in the form of tabulated data, either directly from XFOIL or XFLR5 or via other means.

$$c_l(\alpha, Re) \quad \text{and} \quad c_d(\alpha, Re) \quad (5.7)$$

<sup>1</sup>There is no contribution in  $y_{TTP}$  as the BEMT represents the blade through 2D elements

The elemental lift and drag can now be obtained through the known aerodynamic coefficients, alongside the dynamic pressure and the element's chord.

$$d\mathcal{L} = \frac{1}{2}\rho W^2 c_{l_i} dy \quad d\mathcal{D} = \frac{1}{2}\rho W^2 c_{d_i} dy \quad (5.8)$$

Finally, the more useful notation consist in replacing the forces in the axes of the rotor and derive the thrust ( $\mathcal{T}$ ), torque ( $\mathcal{Q}$ ) and power ( $\mathcal{P}$ ) contribution of the element

$$d\mathcal{T} = N_b(d\mathcal{L} \sin \phi - d\mathcal{D} \cos \phi) \quad (5.9)$$

$$d\mathcal{Q} = N_b(d\mathcal{L} \cos \phi + d\mathcal{D} \sin \phi)y \quad (5.10)$$

$$d\mathcal{P} = N_b(d\mathcal{L} \cos \phi + d\mathcal{D} \sin \phi)y\Omega \quad (5.11)$$

## 5.4 Momentum theory

The momentum theory considers the rotor as an infinitesimally thin actuator disk. Following this theory, we consider that

- the flow velocity and pressure are constant on every section normal to the throughflow in the rotor stream tube;
- the rotor induces an abrupt change of the flow conditions by causing a pressure jump across the disk.

Following the Blade Element Theory, the overall control volume can be further refined in infinitesimal concentrically arranged annuli whose area is expressed by  $dA = 2\pi y dy$ . The mass flow rate through any individual annulus is then given by:

$$\dot{m} = \rho(V + v_i)dA = \rho(V + v_i)2\pi y dy \quad (5.12)$$

Furthermore, it can be showed that axial induced velocity at the disk is equal to half the slipstream velocity in the far wake:  $v_i = \frac{v_w}{2}$ . The thrust can then be expressed using the momentum balance in the axial direction as:

$$\begin{aligned} d\mathcal{T} &= \dot{m}(V_d - V_u) \\ &= \dot{m}v_w \\ &= \dot{m}(2v_i) \\ &= 4\pi\rho(V + v_i)v_i y dy \end{aligned} \quad (5.13)$$

The ideal power can be calculated similarly as it is expressed by the product of the thrust and the induced velocity:

$$\begin{aligned} d\mathcal{P} &= d\mathcal{T}v_i \\ &= 4\pi\rho(V + v_i)v_i^2 y dy \end{aligned} \tag{5.14}$$

FiXme: torque equation

FiXme!

## 5.5 Solvers

The essence of the Blade Element Momentum Theory, is now to combine the Blade Element equations for the thrust and power (5.9, 5.11) and the corresponding momentum equations (5.13, 5.14). This newly formed system can then be solved for the inflow angle and the induced velocity at the disk.

Unfortunately, solving such a non-linear system is not trivial. ROTARE does that by implementing four different solvers, all based on the same set of initial equations. These solvers differ by introducing some additional assumptions or by modifying the nonlinear system of equations (mostly the momentum equations) in order to simplify its formulation or ease the convergence of its solution. A quick comparison of the solvers is presented in Table 5.2.

**Table 5.2.:** Comparison ROTARE's solvers

	<b>Leishman</b>	<b>indFact</b>	<b>indVel</b>	<b>Stahlhut</b>
Assumptions	Small angles $V \ll U$	-	-	-
Applications	Drag $\ll$ Lift Hover/idle and slow axial flow	Any <sup>2</sup>	Any	Any
Convergence	Guaranteed	Medium	Medium	Easy
CPU time	Fastest	High	High	Medium

<sup>2</sup>Technically it does not work with idle/hovering rotors directly. This limitation has been circumvented in ROTARE.

### 5.5.1 Leishman solver

This solver is based on the methodology described by Leishman in [Lei06]. This solver makes some strong assumptions on the flow and the operation of the rotor in order to linearize the system as much as possible. These assumptions correspond to a rotor lightly loaded, which is linked to small angles approximations. This solver is therefore only suitable for hovering rotors and slow axial flows.

#### Assumptions

The following three assumptions are made in order to linearize the system:

1. In-plane induced velocities are negligible:

$$W \approx U \quad u_i \approx 0 \quad (5.15)$$

2. Induced angle  $\phi$  is small:

$$\phi \approx \frac{V}{U} \quad \sin \phi \approx \phi \quad \cos \phi \approx 1 \quad (5.16)$$

3. Drag is much smaller than lift and does not contribute much to the thrust so that:

$$d\mathcal{D} \sin \phi \approx 0 \quad d\mathcal{D} \cos \phi \approx d\mathcal{D} \quad (5.17)$$

4. The lift coefficient is linear:

$$c_l \simeq c_{l,\alpha} \alpha \quad (5.18)$$

Note that the fourth assumption can be pushed further by considering that  $c_{l,\alpha} = 2\pi$  (thin airfoil theory). If the airfoil polars are provided, it is also possible to retrieve the true lift curve slope for more precision. Finally, ROTARE allows to remove this hypothesis by calculating the proper lift coefficient using the polars provided instead of assuming the linear law. However, this requires an additional iterative scheme and extends a bit the calculation time.



## Equations

The complete derivation of the equations is left out of this manual. Please refer to [Lei06] for more details.

The BEMT system formed by (5.9, 5.11, 5.13 and 5.14) can be rewritten by taking into account the assumptions just described.

By introducing some interesting non-dimensional factors and ratios such as the local solidity  $\sigma_l$ , the relative position of the element  $r$  or the inflow ratio,  $\lambda$ ; the thrust equation can be rewritten as [FiXme: Verify system and solution](#)

FiXme!

$$\lambda^2 + \left( \frac{\sigma_l c_{l,\alpha}}{4F} r - \lambda_\infty \right) \lambda - \frac{\sigma_l c_{l,\alpha}}{4F} (\beta) r^2 = 0 \quad (5.19)$$

where  $F$  is the Prandtl tip-loss factor (see Section 6.1). This equation is a quadratic expression for the inflow ratio  $\lambda$  whose solution is:

$$\lambda(r, \lambda_\infty) = \sqrt{\left( \frac{\sigma_l c_{l,\alpha}}{8F} r - \frac{\lambda_\infty}{2} \right)^2 + \frac{\sigma_l c_{l,\alpha}}{4F} (\beta) r^2} - \left( \frac{\sigma_l c_{l,\alpha}}{8F} r - \frac{\lambda_\infty}{2} \right) \quad (5.20)$$

If the linear lift coefficient assumption is removed, then the inflow equations become: [FiXme: Verify system and solution](#)

FiXme!

$$\lambda(r, \lambda_\infty) = \frac{\lambda_\infty}{2} + \sqrt{\left( \frac{\sigma_l c_l r^2}{4F} - \frac{\lambda_\infty^2}{4} \right)} \quad (5.21)$$

### 5.5.2 Induction factor

This solver is commonly used for the study of propellers or wind turbines. It does not rely on additional assumptions, but rather on a reformulation of the momentum equations in terms of induction factors. This formulation lightens a bit the expressions and has the benefit to be quite intuitive. Its main drawback is that the new equations are not directly compatible with the analysis of rotors at zero external velocity (such as helicopter in hover or propellers in idle).

The equations are derived by first defining the axial and tangential (swirl) induction factors:

$$V = (1 + a)V_\infty \quad (5.22)$$

$$U = (1 - b)\Omega r \quad (5.23)$$

and then inject them in the momentum equations (5.13, ??).

As it can be seen directly in (5.22), this formulation falls down when the free stream velocity,  $V_\infty$ , is zero. To circumvent that limitation, ROTARE keeps the original form of (5.13) when the external velocity is zero but still uses (5.23) in the torque equation.

Using the inflow factors, the thrust equations becomes

$$\begin{aligned} d\mathcal{T} &= 4\pi\rho(V + v_i)v_i y dy \\ &= 4\pi\rho(1 + a)aV_\infty^2 y dy \end{aligned} \quad (5.24)$$

and the torque equation becomes

$$\begin{aligned} d\mathcal{Q} &= 2\pi y \rho (V + v_i) (2b\Omega y) y dy \\ &= 4\pi y^3 \rho (1 + a) V_\infty b \Omega dy \end{aligned} \quad (5.25)$$

### Resolution

The system made of the Blade Element equations and the new form of the momentum equations is then solved using MATLAB's `solve` function in order to determine the value of both induction factors. The solution is initialized with an axial inflow factor of 0.01 and a tangential induction factor of 0 (no swirl).

### 5.5.3 Induced velocities

This solver is a pure resolution of the initial system, without any major rewrite or additional assumptions. It is a bit more formal, but has the benefit of being directly usable in any flow condition.

### Resolution

The system is solved using MATLAB's `solve` function in order to determine directly the induced velocities at the rotor disk. The solution is initialized by considering that the axial induced velocity is 0.01 m/s and the tangential component is 0.

#### 5.5.4 Stahlhut solver

This solver relies on a complete rewriting of the system in a single nonlinear transcendental equation. This improves significantly the convergence of the system and facilitate the determination of a solution. The only drawback is that the equation formed is much more complex and less intuitive than the ones at the base of the system.

### 5.6 Coaxial rotors

The analysis of coaxial rotors is currently supported for either hovering/idle rotors or axial flows only.

In that cases, the first rotor is evaluated exactly as if it was in isolation. The axial and swirl velocity induced by the first rotor as then passed to the second rotor as inlet velocity. The contraction of the wake is then calculated based on the distance between the two rotors and the velocity profile is then adapted accordingly. The second rotor is then evaluated as if it was in isolation as well, but with the new inlet velocity profile.

### 5.7 Oblique flows

ROTARE currently implements oblique flows for single isolated rotors only. The analysis of such flows for multi-rotor systems involves the determination of the actual wake geometry. This is commonly done through other methods (vortex sheets), which is currently out of the scope of this software.

For oblique flows, the solution of the blade elements will depend on the azimuthal position of the blade. It is therefore required to calculate the forces for all azimuthal positions and then integrate around the complete circle.

## Extensions and corrections

### 6.1 Tip/hub loss

# List of Corrections

While ROTARE does not fully support coaxial rotors or oblique flows yet, the current documentation is written as if it was already the case. . . . .	21
Add schematics . . . . .	21
todo . . . . .	22
todo . . . . .	22
todo . . . . .	22
torque equation . . . . .	25
Verify system and solution . . . . .	27
Verify system and solution . . . . .	27