UNIVERSITY OF PADOVA
FACOLTY OF ENGINEERING

—

UNIVERSITY OF GLASGOW

—

MSc DEGREE IN AUTOMATION ENGINEERING

WIND TURBINE: VARIABLE SPEED FOR INDUCTION GENERATOR

Supervisor (Glasgow): Prof. CALUM COSSAR
Supervisor (Padova): Prof. SILVERIO BOLOGNANI

Senior year student: DANIELE TOMASELLO

ACADEMIC YEAR 2011-2012

to my wife Diane

and to my son Samuel

“ ... fatti non foste a viver come bruti, ma per seguire virtute e canoscenza ... ”

Dante Alighieri, Divina Commedia, Inferno canto XXVI, 119-120

Contents

Sommario

Questo progetto di tesi è stato svolto presso la "University of Glasgow" sotto la supervisione del Professor Calum Cossar, e in associazione con "GAIA Wind", un' azienda produttrice di turbine eoliche di piccole dimensioni. Tramite il Professor Silverio Bolognani, supervisore nella sede padovana, è stato mantenuto il contatto con l' "Università degli studi di Padova".

L' OBIETTIVO di questo progetto è stato duplice: in primis quello di implementare una strategia per il **funzionamento a velocità variabile** di una piccola turbina eolica (-wind turbine- WT) di 11KW che utilizza come generatore una Macchina elettrica Asincrona (Generatore ad Induzione); in seguito quello di creare un **simulatore di laboratorio riscalato** su cui poter eseguire dei test di validazione dell' analisi teorica. Posso con orgoglio affermare che entrambi gli obiettivi sono stati raggiunti.

Il lavoro è iniziato con l' analisi del comportamento del vento, della struttura di una generica turbina, e delle leggi che regolano la **conversione dell' energia eolica** in energia elettrica. Ciò mi ha permesso di prendere dimestichezza con le variabili in gioco, facendomi subito notare che la coppia (T) e la potenza generata (P) sono entrambe funzioni della velocità del vento (v) e della velocità di rotazione (ω sulle pale e Ω sul generatore). Su questo gioca un ruolo fondamentale il **coefficiente di potenza** (C_p) espresso solitamente in funzione del "rapporto tra la velocità periferica delle pale e la velocità del vento libero" (-tip speed ratio- λ). Questo coefficiente è quindi stato ricavato da dati forniti dalla GAIA Wind, al fine di poter effettuare simulazioni il quanto più possibile realistiche. E

di conseguenza è stato determinato anche il coefficiente di coppia (C_t), legato a quello di potenza.

Con le equazioni fondamentali ricavate sono state realizzate delle prime figure eloquenti su come varia la potenza e il coefficiente di potenza al variare della velocità del vento e della velocità di rotazione. E' stato subito chiaro che il funzionamento a velocità di rotazione fissa di GAIA Wind (e di tutte le turbine di piccole dimensioni) risulta molto limitativo.

Si è quindi passati allo studio del **Generatore ad Induzione** (-induction generator- IG), passando dalle equazioni generali, alla classica rappresentazione della curva di coppia in funzione dello scorrimento (le variabili vengono mostrate positive nel funzionamento da generatore). Si è quindi considerato il **controllo** a rapporto tensione/frequenza **(V/Hz) costante**, mostrando come questo generi una famiglia di curve traslate.

Le conoscenze acquisite fino a questo punto sono state unite nell' intento di analizzare la **strategia** per passare dall' attuale funzionamento a velocità fissa della turbina ($\omega = 56[rpm]$) al **funzionamento a velocità di rotazione variabile**. E' stato fatto riferimento alla tecnica nota come "inseguimento del punto di massima potenza" (-maximum power point tracking- MPPT), che si prefigge come obiettivo quello di estrarre dal vento la massima potenza possibile, modificando la velocità di rotazione in una finestra di valori plausibili (compresa, nel caso specifico, tra 42 e 84 [rpm]). A tal fine è stato scelto di cambiare il rapporto di riduzione del gear-box, passando da $18 \div 1$ a $12 \div 1$. Il MPPT è stato valutato essere possibile grazie al controllo V/Hz costante, dotando la turbina eolica di un Inverter controllato. Per la gestione della turbina in tutto il range di funzionamento è stato ipotizzato l' utilizzo di una macchina a stati composta da: "Start", "MPPT", "frequenza fissa 50 Hz", e "potenza nominale".

A questo punto sono state implementate delle **simulazioni Matlab (Simulink)** al fine di anticipare alcuni test e ottenere informazioni utili, durante l' implementazione del simulatore in laboratorio. Lo schema Simulink realizzato era composto

di tre blocchi principali. Uno rappresentava il "simulatore del vento", dove in funzione della velocità del vento e della velocità di rotazione, veniva calcolata la coppia generata dal vento sulle pale. Un secondo blocco rappresentava il "generatore ad induzione" che realizzava la macchina a stati sopra descritta, fornendo come coppia richiesta un valore proporzionale alla differenza tra la velocità di rotazione e la velocità sincrona di funzionamento. Per finire il blocco "meccanico" attuava il cambiamento della velocità di rotazione in funzione della differenza tra la coppia generata dal vento e quella richiesta dal generatore ad induzione.

Con questo schema di simulazione viene presentato un risultato esemplificativo del buon funzionamento della tecnica MPPT implementata e vengono accennate quelle che sono le problematiche riscontrate soprattutto nel passaggio dal MPPT alla "potenza nominale" per improvvisi incrementi della velocità del vento.

Segue la parte implementativa del progetto, dove è stato totalmente realizzato in **laboratorio** un **simulatore riscalato della GAIA Wind turbine 11 [KW]** per poter eseguire i test di quanto analizzato teoricamente. La parte essenziale è stata la connessione del Generatore ad Induzione (IG) con un motore (-wind emulator- WE) che simulava la coppia generata dal vento. Per far ciò si è utilizzato un motore sincrono a magneti permanenti in alternata (-permanent magnet AC- PMAC). Queste due macchine elettriche sono state collegate tra di loro attraverso uno strumento di misura (di coppia e di velocità di rotazione), ed un disco inerziale metallico. Questo disco inerziale è stato necessario per simulare il valore elevato dell' inerzia delle pale di una turbina eolica ed è stato ricavato accuratamente utilizzando una stima del valore reale, riscalato nel nostro sistema in laboratorio.

Entrambe le macchine elettriche sono state connesse al rispettivo sistema di controllo. Ognuno dei due era comunemente composto da laptop, controllore FCIV (concesso dal SPEED Laboratory), Inverter, ed Alimentatore. In aggiunta il controllore del generatore ad induzione utilizzava uno stadio a chopper per dissipare la potenza generata.

Sul simulatore di laboratorio così creato sono state infine evidenziate le relazioni tra le velocità di rotazione, le coppie, e le potenze di laboratorio e quelle

ottenibili nel caso reale sulla vera turbina eolica.

Una notevole parte del lavoro è stata dedicata alla realizzazione di un' **interfaccia Visual Basic** nel laptop, e alla scrittura del **programma "C"** per il funzionamento del controllore FCIV. Ciò è stato fatto sia nel lato relativo al Generatore ad Induzione, sia in quello relativo al PMAC simulatore del vento: nel primo tutto ciò è stato totalmente implementato, mentre nel secondo sono state integrate le parti desiderate, sia nelle interfacce Visual Basic, sia nei programmi "C" già esistenti per altri scopi.

Infine vengono presentati i **risultati** ottenuti per i test in steady state del funzionamento a velocità di rotazione variabile, punto di partenza per la buona comprensione del comportamento di una turbina eolica e per i futuri test dinamici implementabili.

Il primo test presentato mostra il comportamneto della WT al variare della velocità del vento, fissato il controllo del generatore ad induzione (tensione e frequenza fissi). Il test è stato ripetuto per tre differenti valori di controllo. I risultati rispecchiano quanto visto teoricamente, cioè che bassi valori di tensione e frequenza nel controllo dell' IG sono migliori per basse velocità del vento, mentre valori via via crescenti sono migliori al crescere del vento. Questo ovviamente fino ad un valore limite della velocità del vento, dove si presenta necessario ridurre nuovamente tensione e frequenza del generatore ad induzione per non generare una potenza superiore a quella nominale.

Il secondo test presentato, invece, mostra il comportamento della WT al variare del controllo del generatore ad induzione (V/Hz costante), fissata la velocità del vento. Il test è stato ancora una volta ripetuto per differenti valori del vento, tre dei quali nella finestra di MPPT, e uno superiore. Anche in questo caso i risultati rispecchiano quanto visto teoricamente.

Per avere un termine di paragone il quanto più possibile corretto, si è inoltre eseguito il test del comportamento della WT con funzionamento a velocità di rotazione fissa, come attualmente avviene nella turbina eolica GAIA Wind 11[KW] (riconsiderando il rapporto di riduzione originale $18 \div 1$, anzichè $12 \div 1$).

I risultati dei vari test sono stati riassunti in un' unica tabella, mostrando che i dati di **produzione annuale di energia** (-annual energy production- AEP) ottenuti nelle simulazioni (ovv. tradotti al caso reale) sono simili a quanto riportato nel datasheet della stessa GAIA Wind. Si sono quindi potuti paragonare i risultati del funzionamento a velocità di rotazione fissa, con quello a velocità di rotazione variabile, mostrando l' elevato guadagno ottenibile.

Questi risultati sono un punto di partenza molto incoraggiante per le **future analisi dinamiche**, dove la mancata regolazione dell' angolo delle pale e l' impossibilità dell' uso del freno nel normale funzionamento, introdurranno delle problematiche, alcune delle quali già accennate in questo progetto. Queste problematiche quasi certamente ridurranno il guadagno mostrato, ma questo non deve spaventare, in quanto il punto di partenza è davvero buono.

Abstract

In this project of Thesis I worked with the "Università degli Studi di Padova" and the "University of Glasgow", in association with "Gaia Wind", company that realizes wind turbines with one of its plants in Glasgow.

Target of this project was to analyze the Wind Turbine (WT) and the conversion of wind energy into electric energy. With this purpose a small turbine 11[KW] with a Induction Generator (IG) was study; at first with fixed-speed functioning, after developing a variable-speed strategy, where the difference of functioning was related to the supply frequency.

In the first part of the thesis the reader is guided through all the most important equations and graphics, allowing to get familiar with such complex topics. Subsequently a Matlab simulation was created to understand the behaviour of the wind turbine and the consequence of a parameter's change during the laboratory rig development.

Finally a Test rig was completely implemented, with a side representing the Turbine (scaled to our components) and the other side emulating the Wind behaviour. With this Rig some laboratory tests are showed to validate the studies.

(a) University of Glasgow

(b) University of Padova

(c) Gaia Wind

Introduction

Figure 1: *Whitelee wind farm in south of Glasgow, 322 MW*

Renewable energy sources become discretely popular in the last decades, due to their abundant in nature, cost-effectiveness and contribution to the climate change goals.

Among all, wind turbine system is the most promising renewable energy technology. Huge wind farms with multi-MW turbines are being realized in every country, like the "Whitelee wind farm" showed in *Figure 1*, visible from the University of Glasgow. Moreover, the small turbines (tens of KW) are getting a great consent and development: Gaia Wind (*Figure 2*) is a factory that produces wind turbines, having one of its plants in Glasgow, with whom I collaborated in this project.

Figure 2: Gaia Wind turbine 11[kW]

In this work I started analyzing the principles of wind turbines, in relation to wind speed, rotational speed, torque, and power. At the same time I studied the working law, the equivalent circuit, and the equations of a squirrel cage Induction Generator (IG), that I used (like Gaia Wind) for the electric power generation. This notions permitted us to examine the Fixed-speed functioning and introduce the Variable-speed for the Maximum Power Point Tracking (MPPT).

I first created a Matlab scheme, representing the Induction Generator, the Wind Emulator (WE) and the mechanical behavior. This made the development of the real Test Rig easier, and permitted me to do some tests in advance, with results easy to understand. Finally I realized the laboratory rig how showed in *Figure 3*.

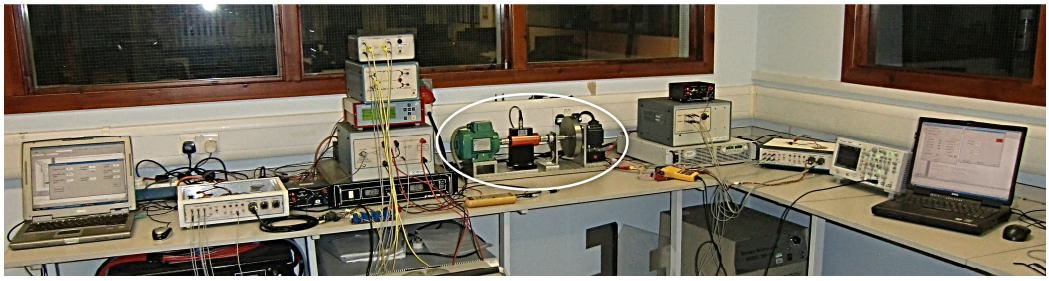


Figure 3: *Laboratory' s test implemented*

The Rig was composed of three parts:

- Induction Generator's controller (left)
- Wind Emulator's controller (right)
- Induction Generator connected with Wind Emulator's motor (central)

The controllers in both sides were composed of PC with control panel interface (in Visual Basic), FCIV Flex Controller from SPEED's Laboratory, power electronic and inverter for PWM. The Wind Emulator was a permanent magnet AC motor (PMAC); it was connected to the Induction Generator through an Inertia disc (that represented the blades of the Wind Turbine) and a "Torque & Rotational Speed's measure instrument".

With this rig I did a lot of tests, emulating what happens in the functioning of a real Gaia Wind turbine (11KW), scaled to our Induction Generator of 180W. I also developed a Variable-speed strategy, showing how much the captured power can be incremented in the same turbine. All these results are presented in the last chapter to validate what studied.

Chapter 1

Wind principles and Turbine overview

In this chapter I will refer to [1], [2], [3] and [4] briefly explaining the wind energy and presenting the composition and the equations of a Wind Turbine.

1.1 Wind's general knowledge

A lot of documentation is available about the argument "wind", but for this work only a general knowledge is needed.

A typical reference is the Wind-Rose like the one in *Figure 1.1*. It represents the distribution of wind direction, added with the information of the wind speed. In the example the wind is very concentrate in the direction NW & NW-W, however not every cases are so lucky: for this reason every Wind Turbine can align itself with the wind flow direction , such principle is known as "yaw system". The example also

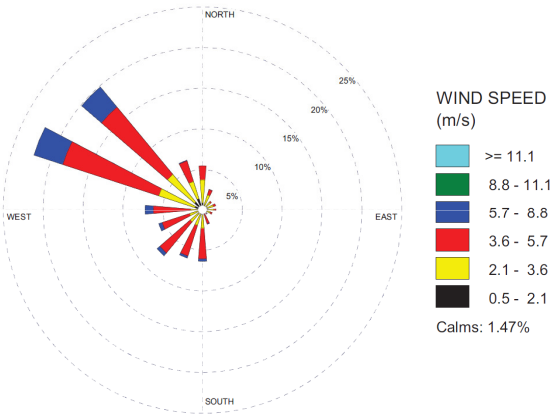


Figure 1.1: Wind rose

shows that the average wind speed is around 4.5[m/s], that is a sufficent, yet low value for a wind turbine installation. The wind flow is really sensible to the presence of physical obstacles, because they create turbulence, making the wind slower and with quickly change of direction. So we will consider the absence of physical obstacles, problem that pertain to the wind turbine's installer.

Another very used figure reference is the Frequency Distribution of the Wind Speed; in which the information on the wind direction is not present, but only the distribution (% or hours/year) of each speed value is showed. *Figure 1.2* presents an example of it.

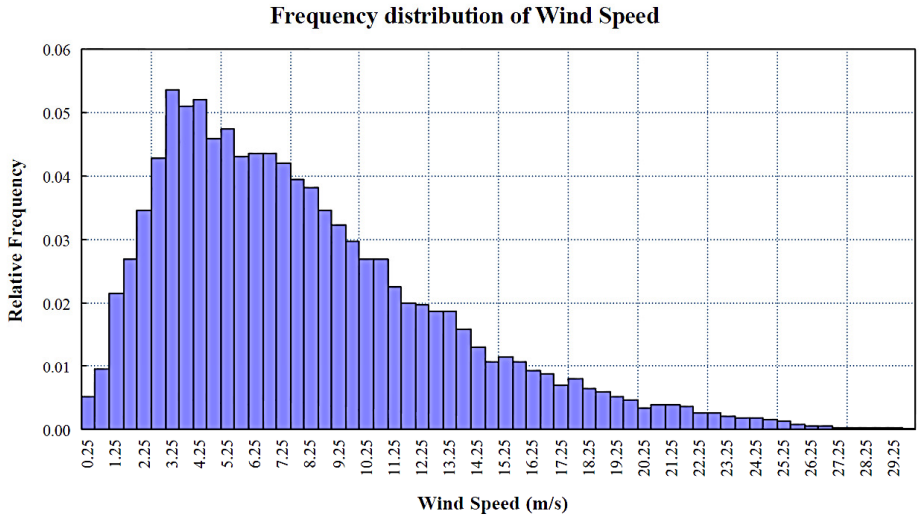


Figure 1.2: *Wind Speed Frequency Distribution*

In this case the average wind speed is higher than the first example and has a good value for a wind turbine installation. We can see that about the 70[%] of the wind speed is concentrate between $2.5 \div 10[m/s]$, with the probability of the small speed that decades quickly, and the probability of the big speed that decade slowly. In fact the Wind Speed Frequency Distribution is usually represented by a WEIBULL distribution, where the probability for the wind speed (v) is a function of two parameters, shape (k) and scale (α):

$$f(\mathbf{v}, \alpha, k) = \frac{k}{\alpha} \cdot \left(\frac{\mathbf{v}}{\alpha}\right)^{k-1} \cdot e^{-\left(\frac{\mathbf{v}}{\alpha}\right)^{k}}; \qquad \mathbf{v} > 0 \tag{1.1}$$

The example in *Figure 1.2* can be represented with a scale parameter α=9.3[m/s] and a shape parameter k=1.7.

From the continuity equation of fluid mechanics, the mass flow of air (dm/dt) through a rotor disk of area ($\pi \cdot R^2$) is a function of air density (ρ), and air velocity (v):

$$\frac{dm}{dt} = \rho \cdot \pi \cdot R^2 \cdot v \tag{1.2}$$

and the kinetic energy per unit time (Power) of the flow is given by:

$$P = \frac{1}{2} \cdot \frac{dm}{dt} \cdot v^2 = \frac{1}{2} \cdot \rho \cdot \pi \cdot R^2 \cdot v^3 \tag{1.3}$$

1.2 Wind Turbine overview

To better explain the functioning of a WT I present its general composition, in every part.

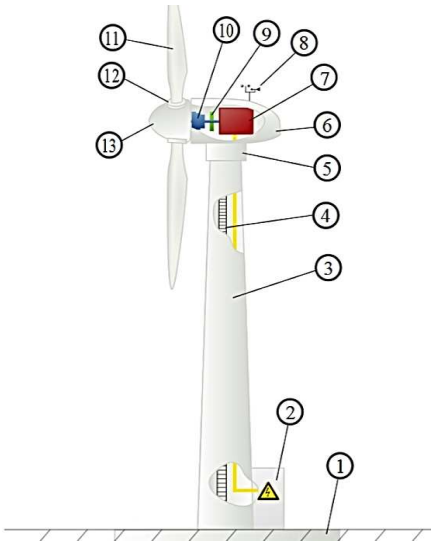


Figure 1.3: Elements

We can see in *Figure 1.3*:

1. foundation
2. control panel
3. tower
4. power cables
5. yaw system
6. nacelle
7. generator
8. anemometer
9. brake
10. gearbox
11. blade
12. pitch system
13. hub

The *control panel* can be composed of control interface, frequency converter and transformer, and can be fixed inside, up to ten meters far from the turbine. The *yaw system* can be free or with active control (usually for the big turbine) and permits the turbine to be aligned with the wind flow. The *nacelle* is the heart of a wind turbine, where generator, brake, and gearbox are allocated. The *generator* is the essential element that translate the mechanical rotation in electrical energy: it can be a permanent magnet or, usually, an induction generator (squirrel cage, wound rotor or double fed). It is connected to a *brake*, needed to reduce the rotational speed in case of general fail or wind over-speed. After the brake there is the *gearbox* that connects the low-speed shaft of the blades with the high-speed shaft of the generator. The *blades* are usually 2 or 3 and have a particular shape to translate the wind flow in mechanical power: they are connected together in the *hub* with a *pitch system* that changes the angle of the blades: it can be fixed or with active control (only in the big turbine) to change the power captured.

The wind turbines are usually divided in two principal groups: the Big and the Small turbines. The Big ones usually have:

1. Power from $\approx 100[KW] - 5[MW]$;
2. tower height $\approx 60 - 100[m]$;
3. variable rotational speed of generator;
4. active control of yaw and pitch system.

The Small ones are characterized by:

1. Power from $\approx 5 - 50[KW]$;
2. tower height $\approx 10 - 30[m]$;
3. (usually) fixed rotational speed of generator;
4. free yaw and fixed pitch

A small turbine will be analyzed, in the particular case the 11[KW] Gaia Wind turbine, that will be entirely presented in the (Sec:1.4).

The principle of the energy translation is showed in *Figure 1.4*:

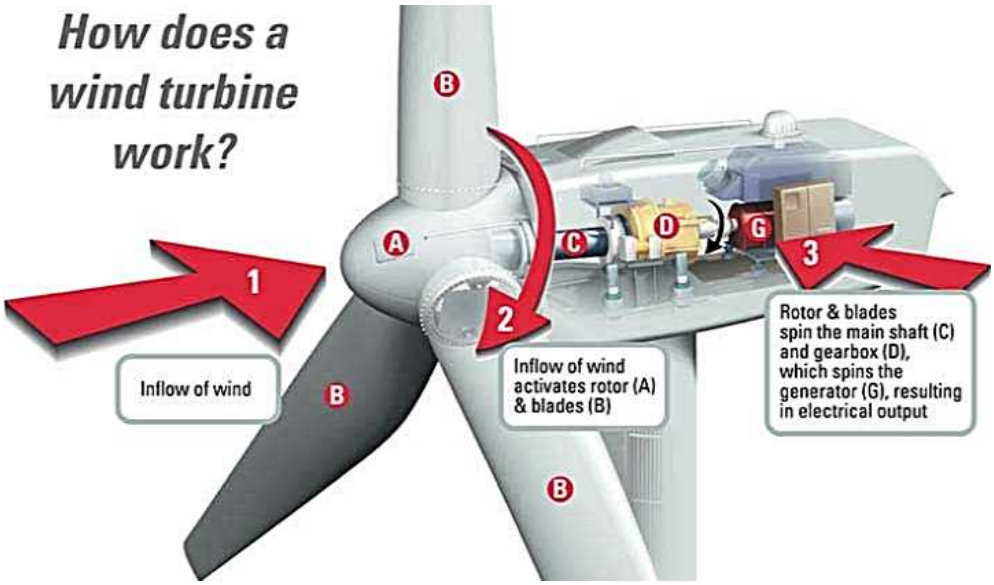


Figure 1.4: *Energy translation*

The figure presents a front-wind functioning, but nothing changes for the back-wind (the Gaia WT functioning). When the wind flows through the turbine,it generates some forces on the blades surface. All these forces are converted by the blades and the hub into the torque of the main shaft that operates with high torque $(T_{blade}) \approx 100 - 1000[Nm]$ and slow rotational speed $(\Omega_{blade}) \approx 10 - 100[rpm]$. This shaft is connected to the gearbox that increases the rotational speed and reduces the torque with a value of gear-ratio (n_r) that can be for example $18 \div 1$. The relations are:

$$\Omega_{gen} = \Omega_{blade} \cdot n_r \tag{1.4}$$

$$T_{gen} = \frac{T_{blade}}{n_r} \tag{1.5}$$

The generator shaft has a torque $(T_{gen}) \approx 10 - 100[Nm]$ and a rotational speed $(\Omega_{gen}) \approx 100 - 1000[rpm]$. The power (P)

$$P = T \cdot \Omega \tag{1.6}$$

remains constant through the gearbox.

Between the generator and the gearbox, the brake is installed. It works in the fast shaft and it is the first emergency step for the secure functioning of a wind turbine. It reduces the rotational speed in the case of over-speed, over-power or general fail.

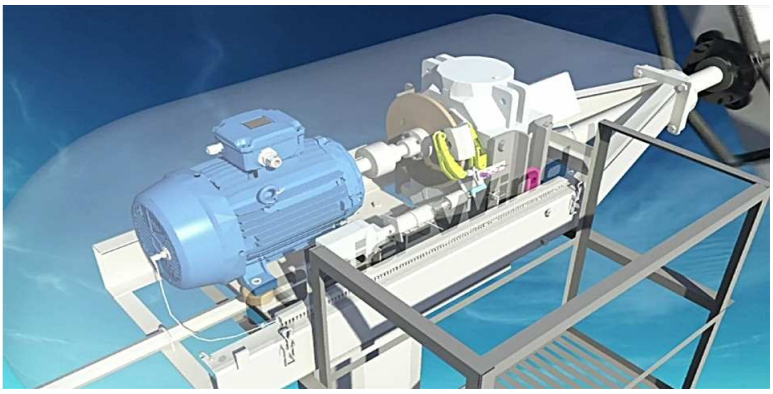


Figure 1.5: *Gaia wind turbine*

Another, drastic emergency step is aerodynamic and is not controlled: it consists in the 90° turning of the top of the blade, with automatic and definitive intervention in case of over-speed. This is enough to brake the aerodynamic shape and reduce considerably the rotational speed. Usually it is never used in the life of a wind turbine, because it means the total loss of control, and after this, a manual action by a technician is needed to re-establish the correct configuration.

1.3 Power and Torque equations

The power captured from the wind is proportional to the area covered by the blade rotation ($\pi \cdot R^2$), the density of the air (ρ), the cubic of the wind speed (v^3) and a power coefficient (C_p):

$$P = 0.5 \cdot \rho \cdot \pi \cdot R^2 \cdot C_p \cdot v^3 \tag{1.7}$$

A similar equation, expressed for the torque is:

$$T = 0.5 \cdot \rho \cdot \pi \cdot R^3 \cdot C_t \cdot v^2 \tag{1.8}$$

where this time C_t is a torque coefficient. The two expressions are the same equation, how will be clear after the explanation of the two coefficients (*eq:*1.10).

The better way to understand the meaning of the power coefficient is to explain it in relation of the tip-speed ratio (λ):

$$\lambda = \frac{\omega \cdot R}{v} \tag{1.9}$$

where (ω [rad/s]) is the rotational speed of the blade's shaft, (R) is the blade radius and (v) is the wind speed. The blade radius is fixed for every different type of wind turbine, so one can imagine that for every fixed wind speed, the increase of ω corresponds to an increase of λ. It is intuitive to think that, fixed v, and slowly increasing ω, C_p would increase to a Maximum value, and consequently decrease for the demanded over-speed. A typical relation is shown in *Figure 1.6*:

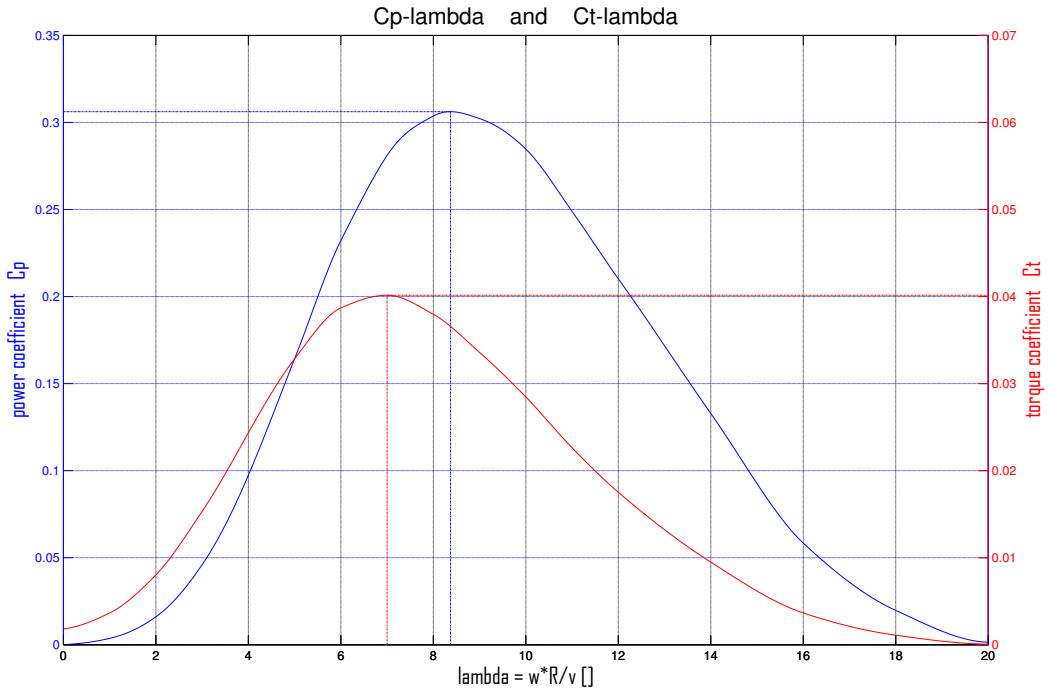


Figure 1.6: *$Cp - \lambda$ and $Ct - \lambda$ relations (emulating GAIA Wind)*

The coefficient C_t is in relation with C_p by:

$$C_p = C_t \cdot \lambda \tag{1.10}$$

C_t has a similar relation figure, but with the maximum value for a smaller λ. Moreover for λ=0 C_p=0 but $C_t > 0$, in fact if $v > 0$ and $\omega = 0 \Rightarrow T > 0$ but P=0. From (*eq:*1.6), (*eq:*1.9) and (*eq:*1.10) one can show the equivalence of the equations (*eq:*1.7) and (*eq:*1.8):

$$P = T \cdot \omega = \underbrace{(0.5 \cdot \rho \cdot \pi \cdot R^3 \cdot C_t \cdot v^2)}_{T} \cdot \omega = \underbrace{(0.5 \cdot \rho \cdot \pi \cdot R^2 \cdot C_p \cdot v^3)}_{P} \cdot \underbrace{(\frac{C_t}{C_p} \cdot \frac{R}{v} \cdot \omega)}_{1}$$

$$in\ fact \quad \Leftrightarrow \quad \frac{C_t}{C_p} \cdot \frac{\omega \cdot R}{v} = 1 \qquad \Leftrightarrow \quad C_p = \frac{\omega \cdot R}{v} \cdot C_t = \lambda \cdot C_t$$

Assuming that a $C_p - \lambda$ relation is true, means that in a WT the behaviour of C_p is the same for multiple of wind and rotational speed (if the ratio remains constant). *Figure 1.7* presents the relation $C_p - \omega$ for 3 different wind speed:

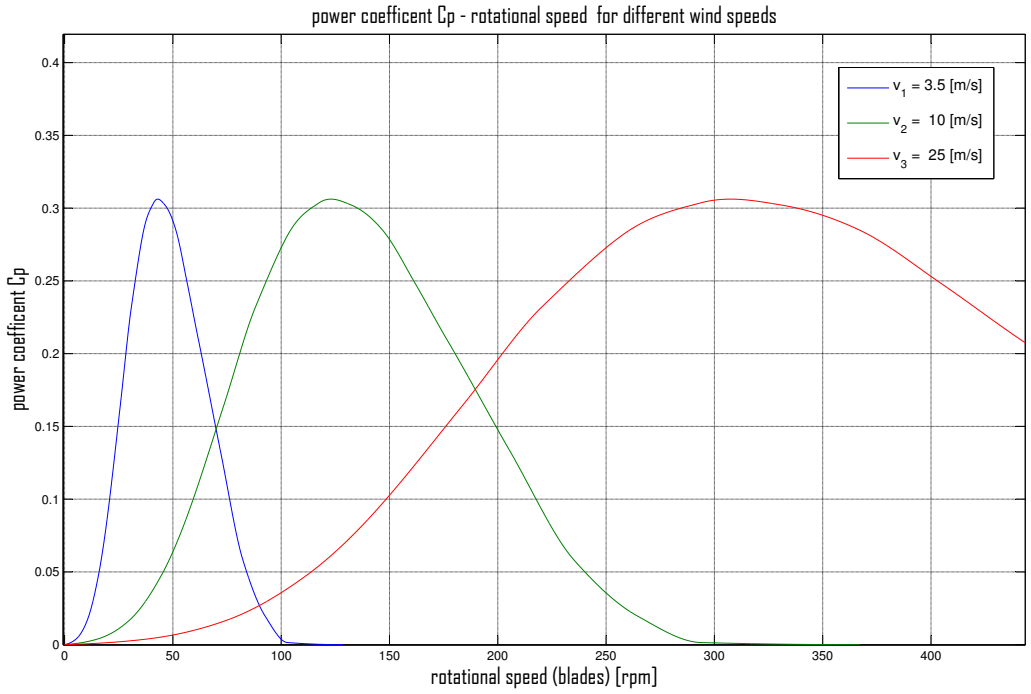


Figure 1.7: *Coefficient Power - Rotational Speed relation (Cp-ω)*

We can see that with a bigger wind speed (v), we have the maximum C_p for a bigger ω and the curve is dilated.

Applying this into the equation (*eq:*1.6) one obtains the power curves showed in *Figure 1.8*:

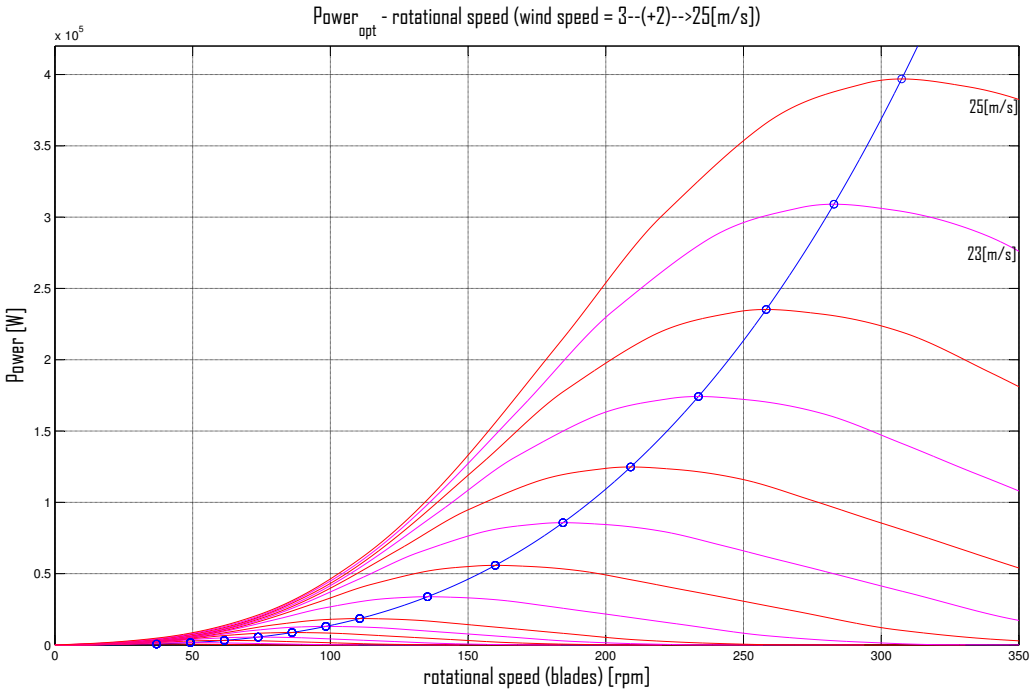


Figure 1.8: *Power (function of the rotational speed) for different wind speeds:* $P(\Omega, \overline{v})$

The Wind Speed curves showed are from 3 to 25 [m/s] with step of 2 [m/s]. The blue curve, instead, is the optimum power that intersect every Wind Speed curve in the point where the power coefficient (C_p) is maximum. We can see that the increase of Wind Speed involve a huge Power available.

It can be useful represent the family of Power curves in a mesh 3D (*Figure 1.9*) where the Power is a function of two variables: Rotational Speed and Wind Speed. In this figure the power available at Fixed-speed functioning (in the example 56[rpm]) is also highlighted to introduce the future observations.

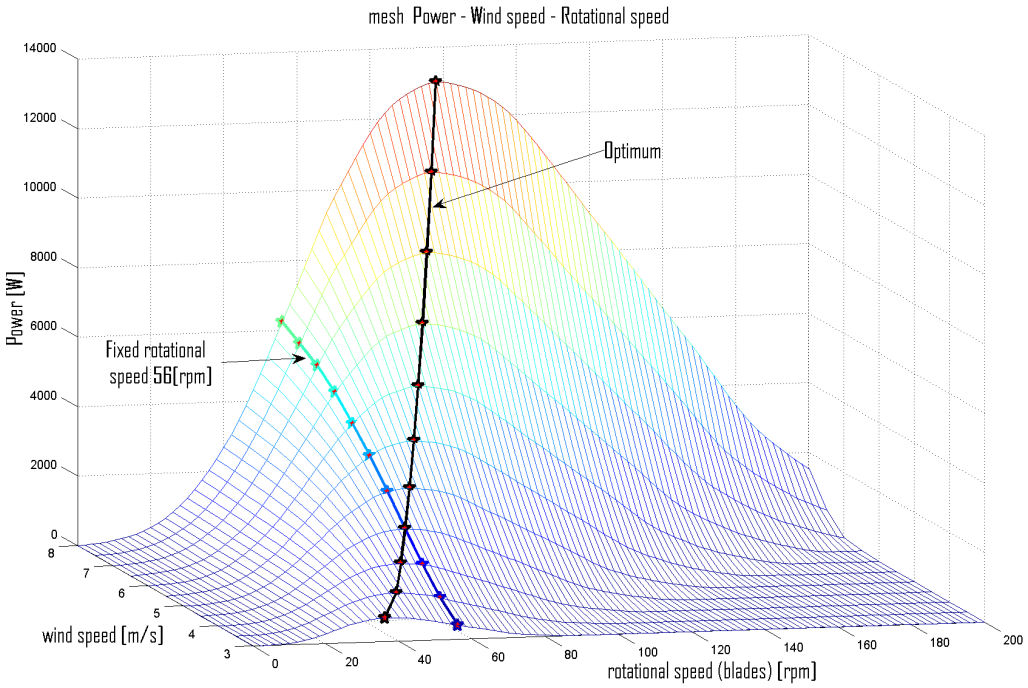


Figure 1.9: *Power for different wind and rotational speed*

1.4 Gaia Wind

Where not differently specified, for all the analysis and consideration I will refer to the Gaia Wind 133-11[KW] turbine. The principal characteristics (MCS data) are:

- nominal power: 11 [KW];
- rotational speed fixed at 56[rpm] (blade shaft);
- blade radius 6.5 [m],area 133 $[m^2]$ (covered by the rotation of the blades);
- gear-ratio $18 \div 1$
- wind speed range $3.5 \div 25$[m/s] (adjustable)

All the data are available in the official datasheet:

Gaia-Wind
GENERATING BETTER VALUE

Gaia-Wind 133-11kW Data Sheet

Annual Energy Production (AEP)*	
Annual Average Wind Speed (measured at hub height)	**Annual Energy Production (AEP)**
4 m/s	16,220 kWh
5 m/s	27,502 kWh
6 m/s	37,959 kWh
7 m/s	46,527 kWh

NOTES:
Figures listed are for 'clean wind sites'. Local topography such as buildings and trees can significantly influence turbine production.

Units shown in domestic electricity bills are in kilowatt-hours (kWh). 1 kWh is roughly equivalent to 1 bar of an electric fire burning for 1 hour.

*Microgeneration Certification Scheme (MCS) data

Noise Profile*	
Sound Power $L_{wd,8m/s}$	88.1 dB(A)
Noise Slope, SdB (dB/m/s)	1.015
Noise penalty	none

Target noise level (8m/s wind)	Distance required
45 dB(A)	60m
40 dB(A)	100m
35 dB(A)	185m

NOTES:
Since the rotor speed of rotation is slow, does not change with wind speed and the blades do not feather or furl, the noise profile of the turbine is very flat making it an exceptionally quiet machine.

*MCS data

Address: Gaia-Wind Ltd., High Croighall Road, Port Dundas, Glasgow G4 9UD United Kingdom, Tel: (+44) 0845 871 4242

www.gaia-wind.com

Key Component Parameters

Twin Blade Rotor	glass fibre, 13m diameter, swept area 133m², mounted on TEETER hub, fixed rotation speed 56 rpm
Gearbox	two stage, gear ratio 18:1, low noise
Generator	11kW, 3 phase, 400V@50Hz (marine grade)
Towers	lattice: 15m 18m monopole: 18m, 27m (hot dip galvanised steel)
Component Weights	nacelle and rotor 900 kg 15m lattice tower 1,556 kg 18m lattice tower 1,955 kg 18m monopole tower 2,511 kg 27m monopole tower 5,275 kg
Standard Presentation	towers: dull grey (galvanised), blade and nacelle cover: grey-white(RAL 9002), reflection free

Operational Parameters

Cut in wind speed (adjustable)	standard setting, 3.5 m/s (5.6 mph)
Shut down wind speed (adjustable)	standard setting, 25 m/s (56 mph)
IEC Turbine class	Conforms to IEC 61400 Class III (suitable for sites with an annual average wind speed up to 7.5 m/s)
Survival Wind Speed	52.5 m/s (117 mph)
Temperature Range	-20°C +50°C
Lifetime and servicing	20 years design life Service once yearly

Control and Monitoring System

Data input and management
Integrated microprocessor with multiple sensor inputs.
Data: wind speed, power, voltages, currents and phase, rpm, vibration and temperature alerts.
LCD display in control box. Can output to local PC or be monitored remotely via internet.

System protection
Base level: Passive stall of blades limits power output.
Second level: Control system activates mechanical brake if:

- Wind speed exceeds 25 m/s
- Abnormal vibration
- Grid disconnected or generator overheats

Third level: Centrifugally activated aerodynamic brakes built into rotor tips as a final safety measure.
Also Manual override button which activates mechanical brake

Certification

UK: Microgeneration Certification Scheme. Certification no. TUV 0002
Denmark: Risø DTU 2009-1

GW-UK 1-611

Address: Gaia-Wind Ltd., High Croighall Road, Port Dundas, Glasgow G4 9UD United Kingdom, Tel: (+44) 0845 871 4242

www.gaia-wind.com

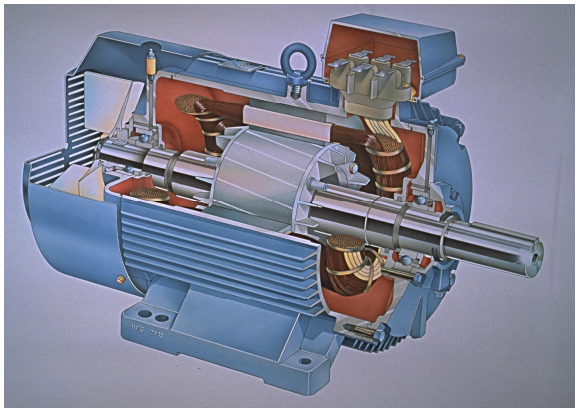
Figure 1.10: *Datasheet GAIA Wind*

Chapter 2

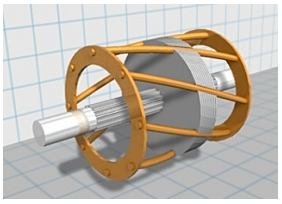
Induction Generator

Every induction (or asynchronous) machine can work like a motor or a generator: at first the power comes from the electric supply, secondly it is taken by the mechanical load. Knowing this aspect I will name the machine *Induction Generator* (for its role) and I will define all the variables with the generator configuration, positive when the machine is generating. This will make easier the analysis in the same figure of IG-Torque and Torque from wind flow. I will silently refer to [5] and to "Lesson's notes of Professor S. Bolognani" from Università degli Studi di Padova.

2.1 Squirrel Cage Induction machine



(a) Induction Generator



(b) Squirrel Cage Rotor

Figure 2.1:

Squirrel cage induction machines are the most commonly used in AC drives, because they are robust, cheap and have low maintenance cost. These advantages make them very attractive for wind power applications. In *Figure 2.1(a)* a classical Induction machine is showed.

STATOR AND ROTOR: The stator of an induction machine consists of poles carrying current to (or from) a magnetic field that penetrates the rotor. To optimize the distribution of the magnetic field, the windings are distributed in slots around the stator, with the magnetic field having the same number of north and south poles (pole pairs). Generating, the stator takes the power from the alternating current till the rotating magnetic field rotates in time with the AC oscillations.

The Squirrel Cage Rotor, showed in *Figure 2.1(b)*, is overall shaped as a cylinder mounted on a shaft. Internally it contains longitudinal conductive bars (usually made of aluminium or copper) set into grooves and connected at both ends by shorting rings forming a cage-like shape. The name is derived from the similarity between this rings-and-bars winding and a squirrel cage. The conductors are often skewed slightly along the length of the rotor to reduce noise and smooth out torque fluctuations that might result at some speeds due to interactions with the pole pieces of the stator. The solid core of the rotor is built with stacks of electrical steel laminations and it serves to carry the magnetic field through the rotor conductors. Because the magnetic field in the rotor is alternating with time, the core uses construction similar to a transformer core to reduce core energy losses.

SYNCHRONOUS SPEED: The theoretical unloaded speed of the induction machine is called *Synchronous speed* and is controlled by the frequency of the supply voltage and the number of pole pairs. It is always an integer fraction of the supply frequency and in revolutions per minute (RPM) is given by:

$$\Omega_o = \frac{f \cdot 60}{p} \tag{2.1}$$

where “f” is the frequency of the AC stator in Hz and “p” is the number of magnetic pole pairs per phase. For example, a small 3-phase motor that has six magnetic poles organized as three opposing pairs 120° apart, each powered by

one phase of the supply current has one pair of poles per phase, which means p = 1, and for a line frequency of 50 Hz the synchronous speed is 3000 RPM.

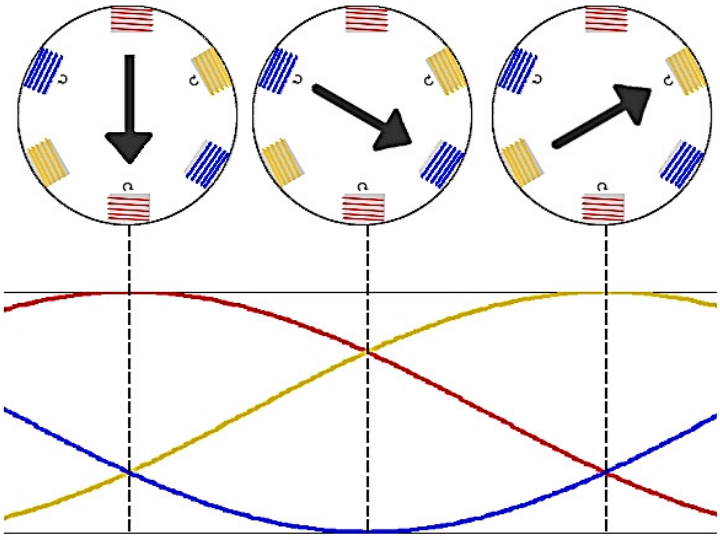


Figure 2.2: *IG phases*

For an increase of pole pairs p = 1 - 2 - 3 - 4 ... , the synchronous speed decreases Ω_o = 3000 - 1500 - 1000 - 750 ... At last for p=1 the mechanical and the electrical period agree; for bigger values, one mechanical period (one rotation) means p electrical periods.

SLIP: The relative motion between the magnetic field in the rotor and the rotation of the rotor induces electric current in the conductive bars. In turn these currents lengthwise in the conductors react with the magnetic field of the motor to produce force acting at a tangent orthogonal to the rotor, resulting in torque to turn (motor) or brake (generator) the shaft. The difference in rotational speed between the rotor and the magnetic field is called *slip* and increases with load. The slip (s) is the rotation rate of the magnetic field, relative to the rotor, divided by the absolute rotation rate of the stator magnetic field:

$$s = \frac{\Omega_m - \Omega_o}{\Omega_o} \qquad (!positive\ generating!) \tag{2.2}$$

where Ω_m is the rotor's rotational speed in rpm. With generating definition, it is zero at synchronous speed, positive generating and negative motoring (-1 when the rotor is stationary). A small slip induces a large current in the rotor and

produces large torque, so for fixed supply frequency the mechanical rotational speed has very small change for the torque variation: for this reason, induction machines are sometimes referred to the asynchronous machines. At full rated load, typical values of slip are 4-6% for small and 1.5-2% for large machines, so induction machines have good speed regulation and are considered constant-speed machines.

GENERATOR-MOTOR, SLIP-TORQUE, BREAKDOWN: For the machine to work as a generator instead of a motor the rotor must be spun just faster than its nameplate speed, this will cause the motor to generate power after building up its residual magnetism. In the Wind Turbine this over-speed is created from the flow of the wind in the blades and transmitted to the machine through the gearbox. The torque exerted in the machine as a function of slip is given by a torque curve.

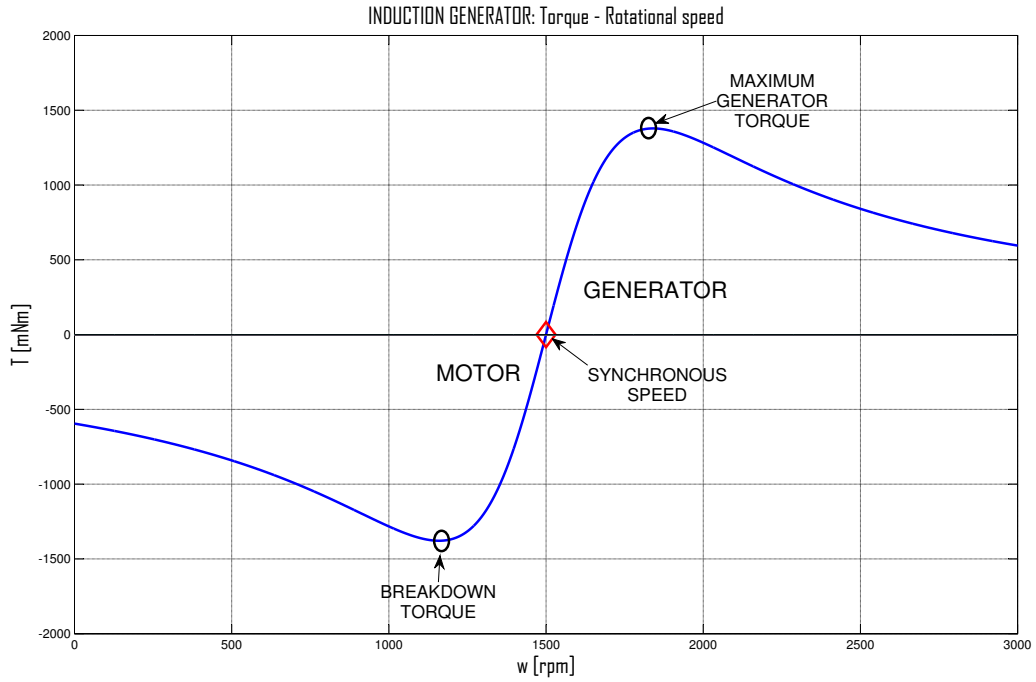


Figure 2.3: *Induction Generator curve (estimated for laboratory's IG)*

Over a generator's normal load range, the torque line is close to a straight line, so the torque is proportional to slip. As the load increases above the rated load,

increases in slip provide less additional torque, so the torque line begins to curve over. Finally at a slip of around 20% the machine reaches its maximum torque ("maximum generator torque" generating, or "breakdown torque" motoring). If the load torque reaches this value, the machine increase quickly and without control his speed if generating, or stall if motoring.

2.2 Working Law, Equivalent Circuits and Equations

I will present the equivalent circuit used in the motor analysis, with the mechanical load expressed by a variable resistance (when the machine work like motor). But in the end, according with the slip definition, I will define the Torque positive generating.

The general equations of an Induction Machine in a general rotating reference system (x) are:

$$\begin{cases} \overline{U}_s^x = R_s \cdot \bar{i}_s^x + \frac{d\bar{\lambda}_s^x}{dt} + j \cdot \Omega_x \cdot \overline{\lambda}_s^x \\ 0 = R_r \cdot \bar{i}_r^x + \frac{d\bar{\lambda}_r^x}{dt} + j \cdot (\Omega_x - \Omega_{me}) \cdot \overline{\lambda}_r^x \end{cases} \tag{2.3}$$

where $\overline{U}_s$ is the stator voltage, R_s and R_r are the stator and rotor resistance, $\bar{i}_s$ and $\bar{i}_r$ are the stator and rotor current, $\overline{\lambda}_s$ and $\overline{\lambda}_r$ are the stator and rotor flux, Ω_{me} is the electro-mechanical rotational speed ($p \cdot \Omega_m$) and Ω_x is the rotational speed of the reference system. The flux is defined:

$$\begin{cases} \overline{\lambda}_s^x = L_s \cdot \bar{i}_s^x + L_M \cdot \bar{i}_r^x \\ \overline{\lambda}_r^x = L_M \cdot \bar{i}_s^x + L_r \cdot \bar{i}_r^x \end{cases} \tag{2.4}$$

Where L_s, L_r and L_M are respectively the inductance of stator, rotor and mutual inductance.

We will use electrical values: (Ω_{me}) electro-mechanical rotational speed, (Ω_s) electrical synchronous speed and (Ω_r) difference speed. It is possible to translate these values in the mechanical reference (Ω_m mechanical rotational speed, Ω_o synchronous speed) knowing that:

$$\Omega_m = \frac{\Omega_{me}}{p}; \qquad \Omega_o = \frac{\Omega_s}{p} \tag{2.5}$$

For the V/Hz control that we will need, a simple equivalent circuit can be represented in a symbolic form by:

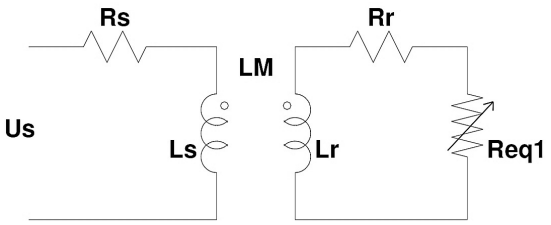


Figure 2.4: *Equivalent circuit 1*

where R_{eq} is a variable resistance that represent the "load", given by:

$$R_{eq1} = R_r \cdot \frac{\Omega_{me}}{\Omega_s - \Omega_{me}} \tag{2.6}$$

Defining :

$$\begin{cases} L_t = L_s - \frac{L_M^2}{Lr} \\ L_p = \frac{L_M^2}{Lr} \end{cases} \tag{2.7}$$

$$R_{rs} = R_r \cdot (\frac{L_M}{L_r})^2 \tag{2.8}$$

$$R_{eq2} = R_r \cdot (\frac{L_M}{L_r})^2 \cdot \frac{\Omega_{me}}{\Omega_s - \Omega_{me}} \tag{2.9}$$

we can represent the scheme :

At last, applying Thevenin on the left side of the circuit, and doing some simplifications I obtain:

where:

$$R_{eq3} = R_r \cdot (\frac{L_M}{L_r})^2 \cdot \frac{\Omega_s}{\Omega_s - \Omega_{me}} \tag{2.10}$$

We arrive so to present the torque equation:

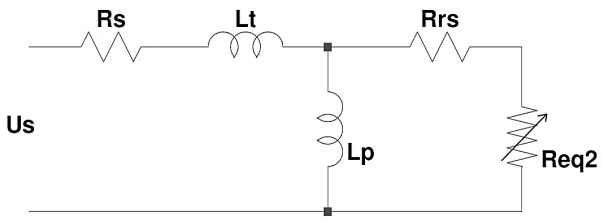


Figure 2.5: *Equivalent circuit 2*

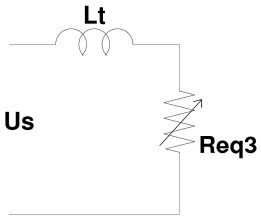


Figure 2.6: *Equivalent circuit 3*

$$T = 3 \cdot p \cdot \left(\frac{U_s}{\Omega_s}\right)^2 \cdot \frac{R_{rs} \cdot \Omega_r}{R_{rs}^2 + \Omega_r^2 \cdot L_t^2} \tag{2.11}$$

where Ω_s is the electrical synchronous speed and Ω_r is the differential speed (that we define positive generating):

$$\Omega_r = \Omega_{me} - \Omega_s \left(= \frac{\Omega_m - \Omega_o}{p}\right) \tag{2.12}$$

Translating these electrical values in mechanical values as explained in (*eq:*2.5) we obtain the torque curve anticipated in the *Figure 2.3*:

In the equations all the Rotational Speeds are [rad/s] values, even if I show [rpm] for visual simplicity.

2.3 V/Hz control

From the equations presented above is clear that changing *simultaneously* the amplitude (U_s, named voltage V) and the frequency (Hz) of the supply, we have

the same curve translated in the rotational speed axis. In fact in (*eq:*2.11) the ratio V/Hz (U_s/Ω_s) remain constant, but change Ω_r for the change of Ω_s in (*eq:*2.12).

For the change of amplitude I will simply use the *Modulation Index* (m_i), a coefficient with values between 0% and 100% multiply for the maximum amplitude.

So for example, changing the IG supply from (f=50[Hz] & m_i=100%) to (25[Hz] & m_i=50%) by steps of (5[Hz] & m_i=10%) we obtain:

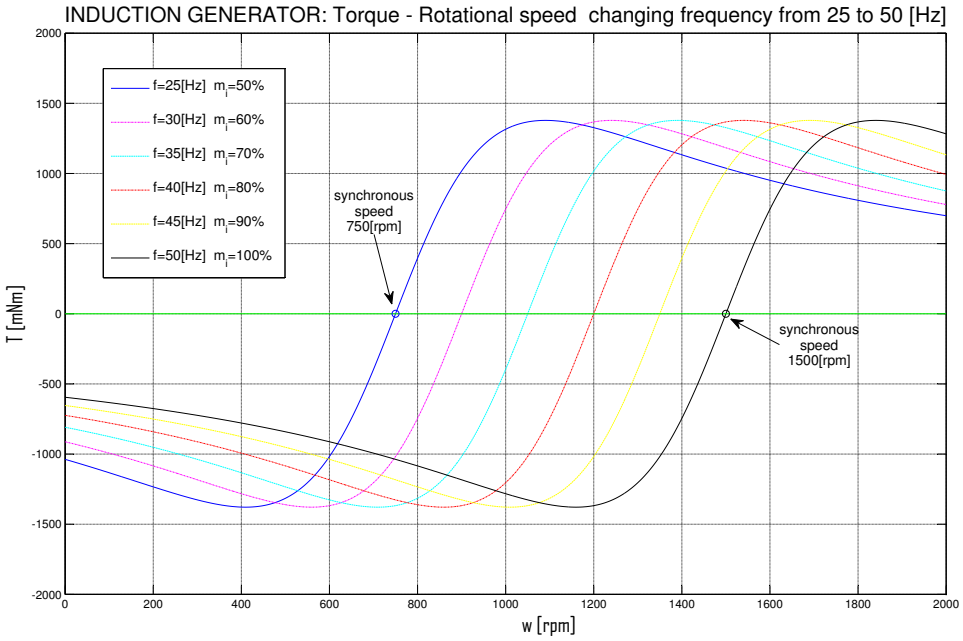


Figure 2.7: *IG multicurves for different V/Hz control value (estimated for laboratory's IG)*

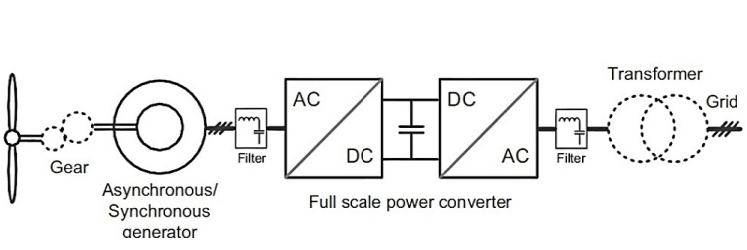
In the figure the curves are estimated for the Induction Generator in the laboratory: Leroy Somer 180[W] with 2 pole pairs. The Induction Generator in the GAIA Wind turbine is a 11[KW] with 3 pole pairs (synchronous speed is at $25[Hz] \rightarrow 500[rpm]$ and at $50[Hz] \rightarrow 1000[rpm]$).

Chapter 3

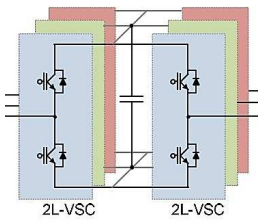
Wind Turbine

In this chapter I will connect as showed in the *Turbine overview* and in *Induction Generator* to present the real functioning of a wind turbine, explaining it in *Fixed Speed* and introducing a strategy for the *Variable Speed functioning* for the *Maximum Power Point Tracking* (MPPT). In the end I will quickly present some Matlab Simulations, useful to introduce in the next chapters the *Laboratory Tests*. I will refer to [6], [7] and [8].

3.1 Functioning description



(a) Functioning scheme of a wind turbine (Def01)



(b) Full scale power converter (Def01)

Figure 3.1:

From many possible interfaces Generator-Grid I will consider the scheme presented in *Figure 3.1(a)*. The generator side and the grid side use two different power converter interfaced by a DC-link, in a configuration presented in *Fig-*

ure 3.1(b) with the name *full scale power converter* (or two level back-to-back voltage source converter).

In the DC-link the voltage and the current are continuous, and the energy is stored in the capacitor. The increase or the decrease of the capacitor's charge (and voltage) is due to the balance of power exchange in the converters of generator and grid side.

Objective of this Thesis project was to analyze the generator side, making maximum the power captured from the blades and sent to the DC-link. The grid faults and the power transmission's problems of the grid side will not be analyzed. So I will speak about blades, blade shaft, gearbox, generator shaft, Induction Generator, IG power converter and DC-link.

3.2 From Fixed to Variable Speed

Reproducing the Fixed Speed functioning, with the scheme in *Figure 3.1(a)* means to use the power converter to re-create the grid voltage: 3 phases 400 $V_{eff-conc}$ at 50 [Hz] shifted by 120°. The wind through the blades increase the mechanical speed (Ω_m) over the synchronous speed (Ω_o), making the slip (s) positive (with generating definition). The sinusoidal current in every phase have a shift, related to the sinusoidal voltage, to create a positive average power (gener. def.). The sum of the 3 phases creates a positive generated power (like positive continuous current from the converter to the capacitor) that increase the DC voltage.

The power generated is indeed related to the wind speed. Reminding that in the actual Fixed-Speed functioning of GAIA Wind the supply frequency is 50[Hz], the pole pairs are p=3, the synchronous speed in the IG shaft is Ω_o=1000[rpm] and the gear ratio is $n_r = 18 \div 1$, we have a Rotational Speed of the blades shaft fixed at $\omega_m \approx 56[rpm]$. So reviewing how showed in *Figure 1.8* we can see in *Figure 3.2*:

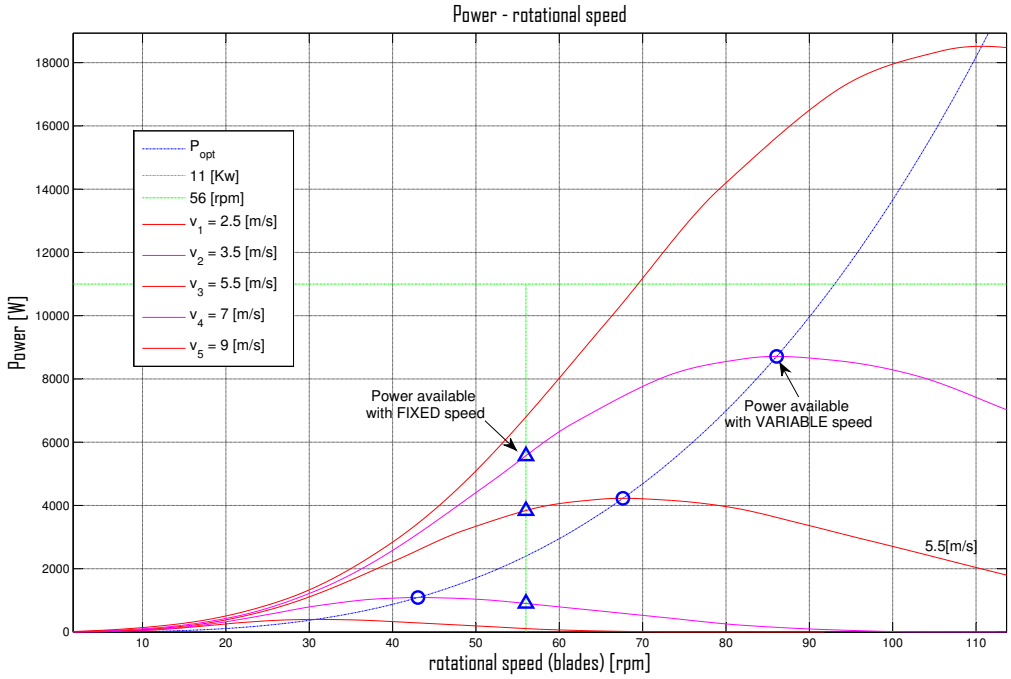


Figure 3.2: *Fixed Speed functioning related to Power available*

It is clear in the figure that one generates the maximum possible power only for the particular wind speed (v=5[m/s]): for greater or smaller wind speeds one generates less power than the optimum, caused respectively by the under-speed and the over-speed demanded to the blades. Another evident thing, for this particular WT, is that the rated power of 11 [KW] is available from the wind speed v = 7.5 [m/s], but at 56 [rpm] the power generated is only 6 [KW]. For all higher wind speeds, the rated power is available but it is not really captured for the over-stall functioning.

It is consequently reasonable to follow the optimum curve during the increase of the wind speed, and when the Power reach the rated value, decrease the speed to remain at 11 [KW] in stall-functioning. This functioning principle will permit the WT to increase the generated power, like showed in figure:

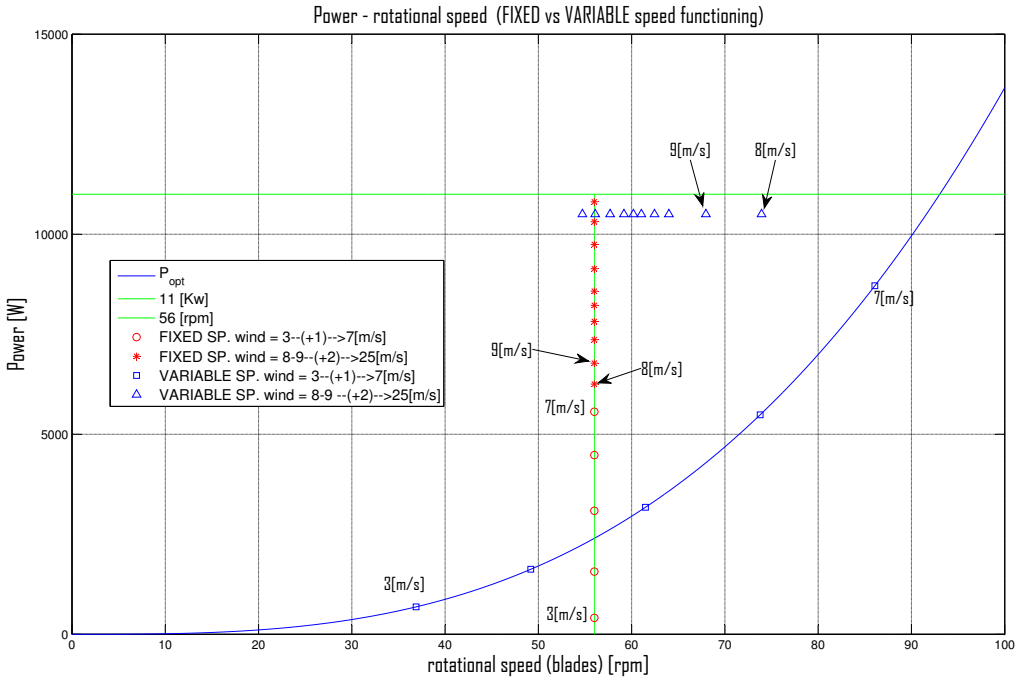


Figure 3.3: *Power available from Fixed-Variable speed functioning*

with the values showed in the table:

	wind speed [m/s]										
	3	4	5	6	7	8	9	11	15	19	25
Fix[W]	410	1570	3090	4480	5560	6250	6780	7360	8220	9140	10810
Var[W]	690	1630	3180	5490	8710	11000	11000	11000	11000	11000	11000

Table 3.1: Fixed-Variable speed table

3.3 Maximum Power Point Tracking

The MPPT is the control technique with objective follow the point of maximum power available by the wind speed (in the limit of the rate power). To this purpose, the frequency of the Induction Generator is changed in a window of possible values: we assume a minimum value $f_{min} = 25[Hz]$ and a maximum value $f_{max} =$

$50[Hz]$. The use of V/Hz (constant) control imposes that the voltage change with the frequency; the modulation index of the supply voltage has therefore the range: $mi_{min} = 50\%$ and $mi_{max} = 100\%$. Some typical values of the couple frequency-modulation index are, as showed in *Figure 2.7*:

$$\begin{pmatrix} f = 25[Hz] \\ m_i = 50[\%] \end{pmatrix} ... \begin{pmatrix} 30[Hz] \\ 60[\%] \end{pmatrix} ... \begin{pmatrix} 45[Hz] \\ 90[\%] \end{pmatrix} ... \begin{pmatrix} 50[Hz] \\ 100[\%] \end{pmatrix} \tag{3.1}$$

where for a variation of $\Delta f = 1$ [Hz] correspond a variation of $\Delta m_i = 2$ [%].

With the actual Gearbox (with ratio $18 \div 1$) the window of Optimum Power for Rotational Speed of the blades would be $31[rpm] < \omega_m < 57[rpm]$: as showed in *Figure 3.2* this limits are lower than the good range. So I chose to reduce the Gear-Ratio to $(12 \div 1)$ obtaining:

- rotational speed range (blade): $42[rpm] < \omega < 84[rpm]$
- wind range for MPPT: $v_{min} \approx 3.5[m/s]$ to $v_{max} \approx 7[m/s]$
- Power range for MPPT: $P_{min} \approx 1100[W]$ to $P_{max} \approx 8700[W]$

From the wind speed $v > 7[m/s]$ the supply frequency can remain constant with $f = 50[Hz]$ to the reaching of rated power $P = 11[KW]$ for $v \approx 7.7[m/s]$. For a further increase of the wind speed we can start to decrease the supply frequency to reduce the synchronous speed and consequently the blade's rotational speed, working in stall region. The maintenance of the rated power is due (*eq:*1.7) to the decrease of C_p.

This strategy was managed with a **State Machine**:

1. Start Turbine;
2. Maximum Power Point Tracking (MPPT);
3. fixed frequency $f = 50[Hz]$;
4. maintenance of Rated Power.

3.4 Matlab (Simulink) Simulation

3.4.1 Scheme

I developed an approximative Matlab simulation to quickly check the results before emulate the turbine in the Laboratory Rig. I used the following scheme (reduced to be easy understood):

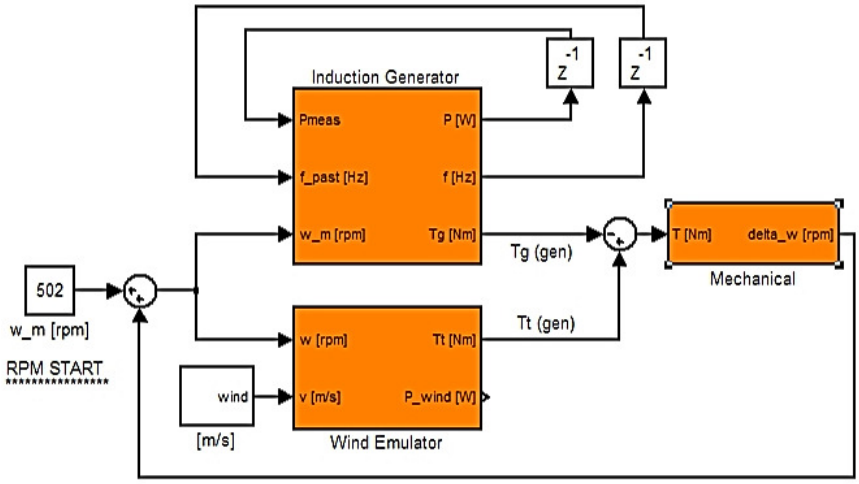


Figure 3.4: *Matlab simulation*

where the Induction Geneartor and the Wind Emulator are present both connected with the Mechanical System that emulate the turbine, translating the difference of torque in decrease or increase of rotational speed. In the real turbine the wind flowing in the blade creates a Torque (that tries to increase the rotational speed), but the Induction Generator creates an opposite Torque (that tries to decrease the rotational speed) due to the power generation: if these 2 torques are balanced nothing happens, otherwise the Rotational Speed will change with a velocity depending on the blades Inertia.

The **Mechanical** block is realized according to the following equation:

$$T_m = J \cdot \frac{d\omega}{dt} \tag{3.2}$$

Where the Inertia is estimated of $1600[Kg \cdot m^2]$ as it will be showed in (*eq:*4.1).

The other gains are needed to translate the Rotational Speed from the generator shaft to the blades shaft, and from [rpm] to [rad/s] (and viceversa).

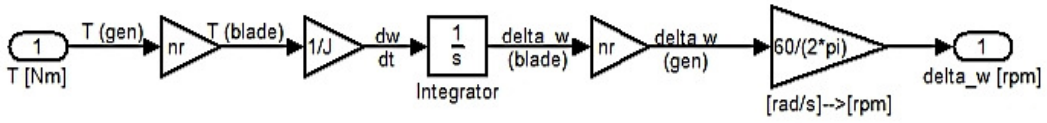
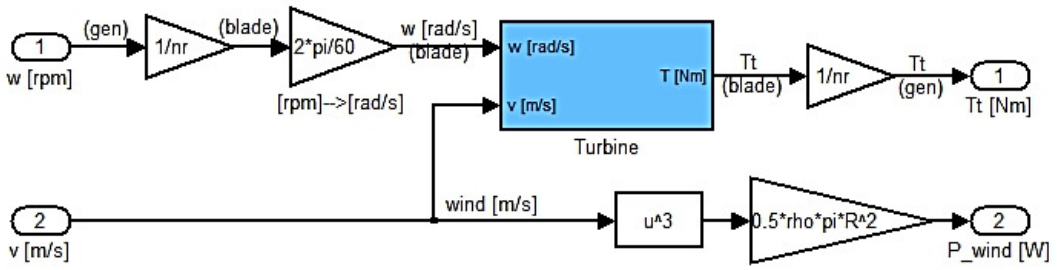
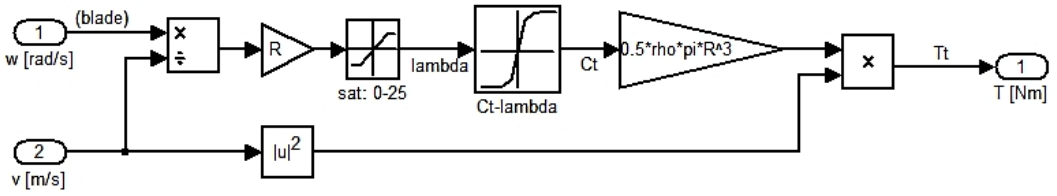


Figure 3.5: *Mechanical system*

The **Wind Emulator** is composed of all the equations seen until now (*eq:*1.7) (*eq:*1.8) (*eq:*1.9) and various gain.



(a) Wind Emulator



(b) WE's subsystem: Turbine

Figure 3.6:

The **Induction Generator** realizes the described State Machine.

In the *START* state the wind rises to the minimum level, the blades are free to increase the rotational speed. From this moment to the rise of the synchronous speed the Induction Machine works like a motor, taking the power from the grid (not generating). But this aspect is not in the interest of this thesis.

In the *MPPT* state the controller drive the IG trying to extract the maximum Power from the wind in the rotational speed range $42[rpm] < \omega < 84[rpm]$

(remember gear-ratio changed to $12 \div 1$).

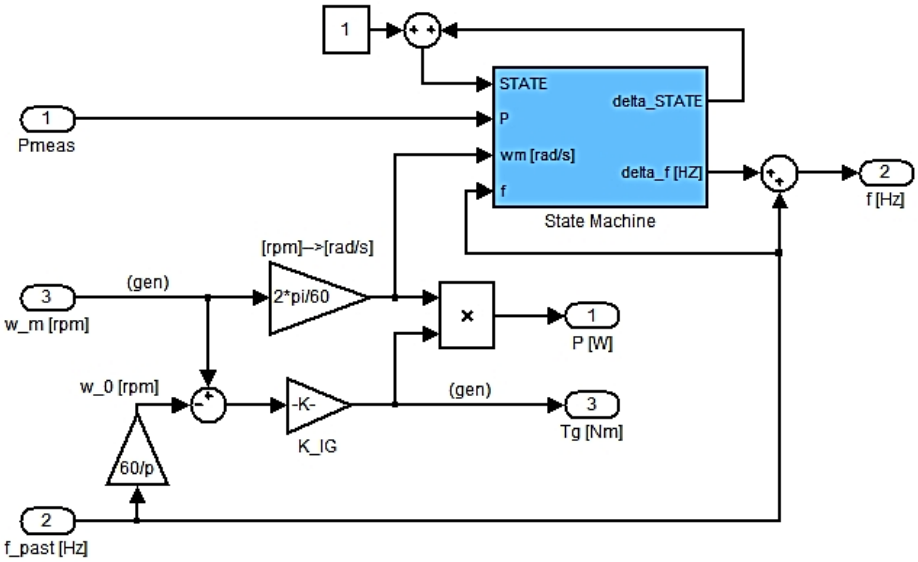


Figure 3.7: *Induction Generator*

It use this technique:

- supposing to be in the optimum point ($P = P_{opt}$, $v = \widehat{v}$, $C_p = Cp_{opt}$, $\lambda = \lambda_{opt}$) it calculate from (*eq:*1.7) the estimated $\widehat{v}$ and after from (*eq:*1.9) the estimated $\widehat{\omega}$.

$$\widehat{v} = \sqrt[3]{\frac{P_{meas}}{0.5 \cdot \rho \cdot \pi \cdot R^2 \cdot Cp_{opt}}}; \qquad \widehat{\omega} = \frac{\lambda_{opt} \cdot \widehat{v}}{R}$$

- if $\widehat{\omega} > \omega$ the controller increases the frequency of the IG of one step (of a prefixed value ex: 0.01[Hz]), otherwise the controller decreases it of one step.

Once the frequency 50[Hz] is reached (rotational speed of blades 84[rpm]) the controller maintains it, working with fixed frequency.

In the *FIXED FREQUENCY* state an increase of wind speed involves an increase of generated power, and a decrease involves a decrease. So the controller checks the generated power: if it is lower than 8700[W] it return to the MPPT state, if it is higher than a threshold (ex: 10.2[KW]) it goes to the Rated Power state.

In the *RATED POWER* state the turbine works in the stall region, taking advantage of the reducing of the power coefficient C_p to remain under the 11[KW]. So for an increase of the wind speed the controller decreases the frequency of the IG, and on the contrary increases the frequency for a decrease of wind speed. The state returns to Fixed Frequency if the controller tries to increases the frequency over 50[Hz]. With the maximum wind speed of v=25[m/s] the rotational speed of the blades is ω_m=56[rpm], the rotational speed of the IG's shaft is Ω_m=670[rpm] (gear ratio $12 \div 1$), which corresponds to a frequency of f=33.5[Hz] (values refered to GAIA Wind's IG with 3 pairs pole).

3.4.2 Results

A functioning example in the MPPT state is the following:

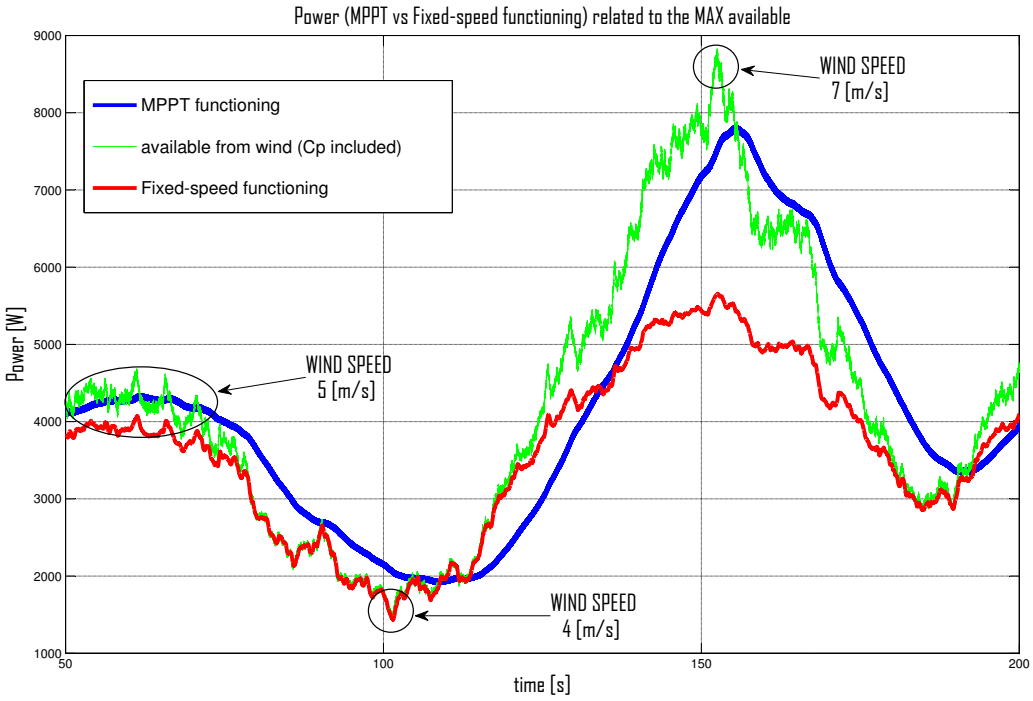


Figure 3.8: *Induction Generator*

I can see that for an increase (or decrease) of the wind speed there is more (or

less) power available: the Variable functioning system follows this available value, but with a delay due to the big inertia of the blades. In fact the blades change slowly the rotational speed, realizing a store of energy. The Fixed functioning, instead, between 5 and 7 [m/s] starts to have a reduced captured power due to the reduction of the power coefficient (C_p) that in the example reach the 65%.

The Variable-Speed functioning showed good results in the MPPT state, but over this (in FIXED FREQUENCY and RATED POWER) there are some problems, due to the nature of the small wind turbine. In the big WTs the pitch regulation is developed to permit (changing the angular of the blades) to reduce the torque and the power captured when the wind reaches high speeds. In the small WTs the pitch of the blades is fixed and to maintain the Power limited for the increase of wind speed over 7-8[m/s] one needs to reduce the power coefficient C_p, therefore the Rotational Speed. Assuming **NOT** to **USE** the **BRAKE** in the normal functioning, to reduce the Rotational Speed means taking out energy from the blade's Inertia, generating OVER-POWER in the IG. This effect creates a particular problem when the wind speed increase from 7 to 9 [m/s]: remembering what showed in *Figure 3.3* with a wind speed of $v = 7[m/s]$ I want a rotational speed of blades $\omega_m = 84[rpm]$, but for $v = 9[m/s]$ I want $\omega_m = 68[rpm]$. This decrease of $\Delta\omega_m = 16[rpm]$ (for $\Delta v = 2[rpm]$) means to have a huge Over-Power due to the energy stored in the blade's Inertia. Trying to slow down ω_m can reduce this effect, but the Power remains too high due to the large value of C_p. For higher wind speeds, between 10 and 25 [m/s], this problem is significantly reduced and this range of 15[m/s] can be managed with the change of 10[rpm]. But the change of frequency have to be very small for every controller's interrupt and the Power can have oscillations.

As just explained is not meant to be a solution, but a warning for the future dynamic analysis.

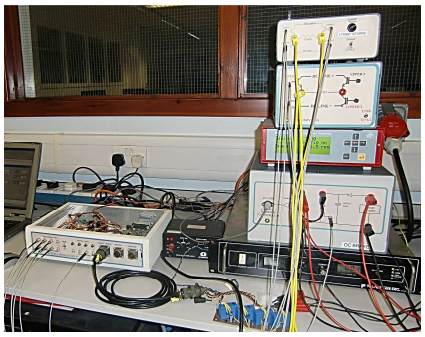
Various solutions can be implemented to solve (or limit) these problems, for example choosing an intermediate gear-ratio $n_r = 15$ *accepting that, not only the problems, but also the gain of the Variable-speed functioning will be reduced.*

Chapter 4

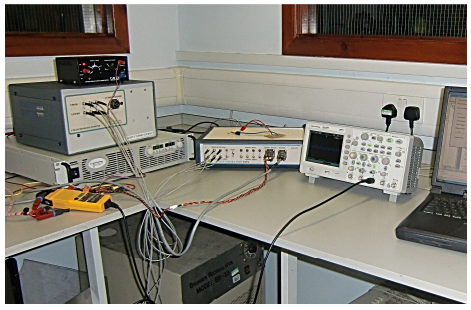
Laboratory description

I start now to describe the elements of the test rig that was realized, trying to remain connected as much as possible to the Gaia Wind turbine 133-11[KW] till now considered. I will refer to [9], [10], [11], [12], [13] and [14].

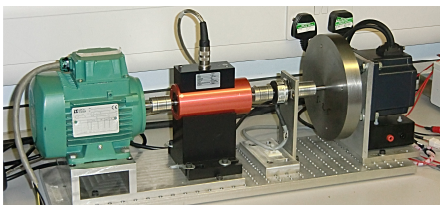
4.1 Component description



(a) Induction Generator's controller



(b) Wind Emulator's controller



(c) IG connected with PMAC Wind Emulator

Figure 4.1:

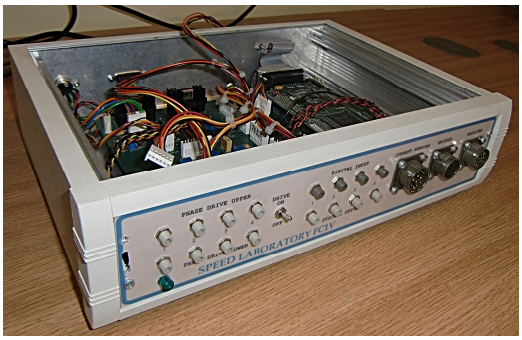
The test rig in the laboratory was composed by 3 parts:

1. Induction Generator's controller *Figure 4.1(a)*
2. Wind Emulator's controller *Figure 4.1(b)*
3. Induction Generator connected to the PMAC wind emulator through measure instrument and Inertia *Figure 4.1(c)*

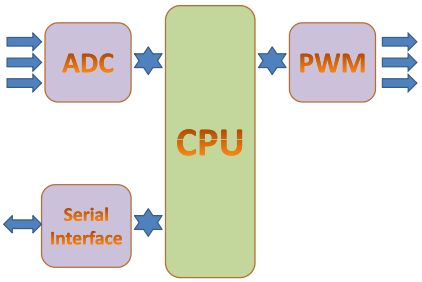
4.1.1 Controllers

As showed in *Figure 4.1(a)* and *Figure 4.1(b)* both the controllers were composed of a laptop, a FCIV-controller, a power supply and a inverter(IGBT).

In the **LAPTOP** a Visual Basic panel was installed, that permitted to interface with the controller, introducing the input and watching the output parameters. It was connected to the **FCIV CONTROLLER** (called also Digital Signal Processor, DSP), showed in *Figure 4.2(a)* and *Figure 4.2(b)*:



(a) FCIV controller



(b) DSP functioning scheme

Figure 4.2:

This was the most important element, because in its CPU ran the control program. The FCIV-controller exchanged its data with the Visual Basic panel by the Serial Interface, using a Serial connection (RS232) with a Command Format of 11 bytes for each communication. In the DSP was also present the ADC (analog-digital converter) for the lecture of the analog inputs and the PWM block that translated the reference signals from the CPU in the 6 output lines to send to the inverter through the optical fibre (3 phases, each with upper and lower signal).

The **INVERTER** translated the signals from the FCIV-controller in power signals that drove the electrical machines (motor & generator).

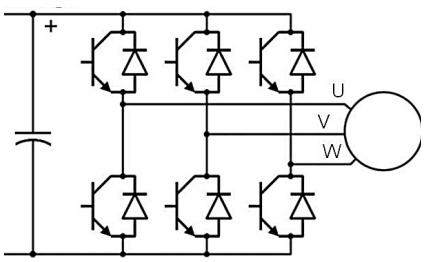


Figure 4.3: Inverter

It was composed of 6 IGBT (insulated gate bipolar transistor) each in parallel with a diode. The IGBT is a three-terminal power semiconductor device, with the simple gate-drive characteristics of the MOSFETs, and high-current and low-saturation-voltage capability of bipolar transistors. The parallel of a diode was necessary for the inverse current. The Inverter drove the electrical machine using a PWM signal, alternating for every interrupt (and in each phase) the supply voltage & 0V, recreating the average voltage desired. For example if the supply voltage was 200V and the interrupt time $100\mu s$ I would have had for every interrupt:

- $t_{200V} = 100\mu s \quad \& \quad t_{0V} = 0\mu s \quad \Rightarrow \quad V_{avg} = 200V$
- $t_{200V} = 80\mu s \quad \& \quad t_{0V} = 20\mu s \quad \Rightarrow \quad V_{avg} = 160V$
- $t_{200V} = 50\mu s \quad \& \quad t_{0V} = 50\mu s \quad \Rightarrow \quad V_{avg} = 100V$
- $t_{200V} = 20\mu s \quad \& \quad t_{0V} = 80\mu s \quad \Rightarrow \quad V_{avg} = 40V$
- $t_{200V} = 0\mu s \quad \& \quad t_{0V} = 100\mu s \quad \Rightarrow \quad V_{avg} = 0V$

The **POWER SUPPLY** in the Emulator side was simply connected to the Inverter, whereas in the Generator side was connected through a chopper stage.

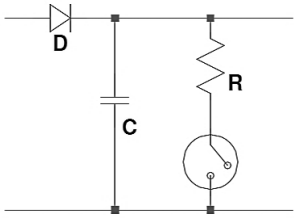


Figure 4.4: Chopper in IG controller

When the Induction machine worked like motor (in the start) in the chopper nothing happened, but when it worked like Generator the chopper had to dissipate the incoming power. Hence it had a diode to isolate the power supply, followed by a capacitor that stored the energy incoming with in parallel a resistor with a driven switch. In the generator functioning the charge of the capacitor increased the voltage, and when it reached a chosen threshold (for ex. 225V) the switch was closed, discharging

the capacitor in the resistor; when the voltage reached a lower threshold (for ex. 210V) the switch was opened again. So the voltage increased and decreased with a average value $V_{avg} \approx 217.5V$

At last, in the sole IG's controller, we had a "Torque & Rotational speed Display" (UMV 2000) to watch these two variables and to send both, as analog values, to the input of the IG controller.

4.1.2 IG and Wind Emulator connection

The connection of Induction Generator and PMAC Wind Emulator is showed in *Figure 4.1(c)*. Obviously both the electrical machines were connected to the respective controller.

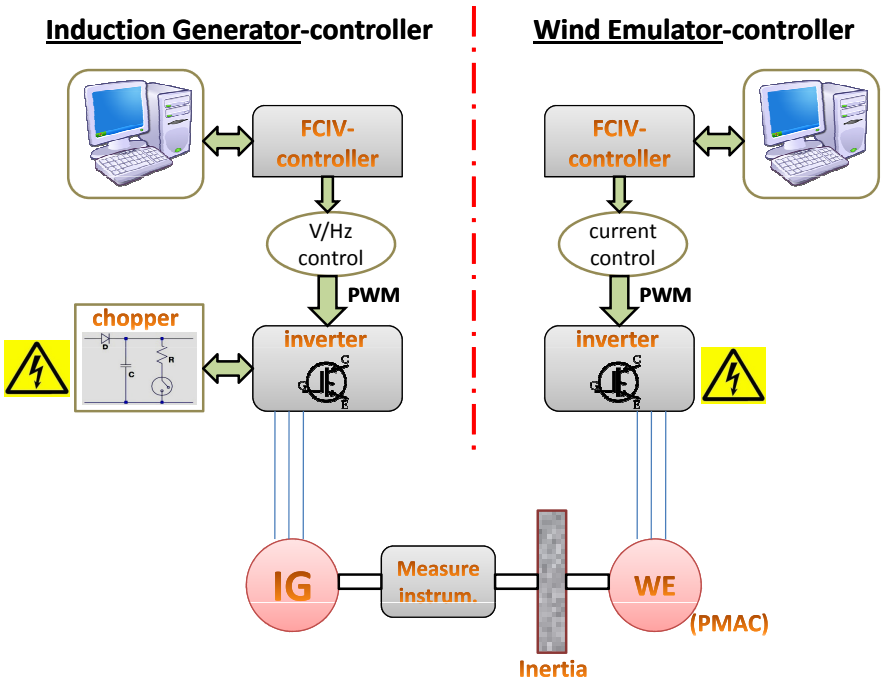


Figure 4.5: *Test rig scheme*

The **INDUCTION GENERATOR** was completely analized in the (*cha:*2). We remember that:

- IG laboratory: 2 polar pairs (p=2) $\Longrightarrow \Omega_o(50Hz) = 1500[rpm]$

- IG GAIA wind: 3 polar pairs (p=3) $\Longrightarrow \Omega_o(50Hz) = 1000[rpm]$

The Induction Generator in the laboratory was a Leroy Somer 180[W] connected in DELTA configuration, where each phase was connected with two point of the inverter, and each inverter's output was connected with two different phases.

The **PMAC WIND EMULATOR** (Permanent Magnet AC) was the motor with the task to emulate the torque created from the wind flow. The torque from wind, (*eq:*1.8), is a function of wind speed and rotational speed (due to torque coefficient C_t). The rotational speed is only the consequence of the interaction between the two machines, and the wind speed is an input from the controller. Therefore the Wind Emulator measured the rotational speed each interrupt and, depending on the wind speed, applied the desired torque T(v,ω) with a current control.

After the IG there was the *TORQUE & ROTATIONAL SPEED MEASURE INSTRUMENT* that sent this two values to the "Torque & Rotational speed Display". This instrument had a precision in the torque measure of 1[mNm] and the limit of 1[Nm]. For the rotational speed it had the precision of 1[rpm] and no limitation problem in our range of work.

Between this instrument and the Wind Emulator there was a **INERTIA DISC** which behaved like the blades inertia in a wind turbine. For choose its value I first estimated the real inertia of the blades in a WT, translating it (with the gear-ratio n_r) in the value for the generator shaft:

$$J_{GAIA-blad} = \sum m \cdot R^2 \approx 2 \cdot 100 \cdot 2.8^2 \approx 1600[Kg \cdot m^2] \tag{4.1}$$

$$J_{GAIA-gen} = \frac{J_{GAIA-blad}}{n_r^2} \approx 11[Kg \cdot m^2] \tag{4.2}$$

After I found the laboratory's value using the ratio of "Rotational speed" and "Maximum torque" between the real and the laboratory's case.

$$\gamma_T = \frac{T_{GAIA}}{T_{lab}} \approx \frac{156}{0.7} \approx 223; \qquad \gamma_\Omega = \frac{\Omega_{lab}}{\Omega_{GAIA}} = 1.5 \tag{4.3}$$

$$\frac{T_{lab}}{J_{lab}} = \frac{d\Omega_{lab}}{dt} = \gamma_\Omega \cdot \frac{d\Omega_{GAIA}}{dt} = \gamma_\Omega \cdot \frac{T_{GAIA}}{J_{GAIA}} \tag{4.4}$$

$$J_{lab} = \frac{1}{\gamma_T} \cdot \frac{1}{\gamma_\Omega} \cdot J_{GAIA-gen} \approx \frac{1}{223} \cdot \frac{1}{1.5} \cdot 11 \approx 0.0329[Kg \cdot m^2] \tag{4.5}$$

It was realized by a disc with radius r=9[cm] and mass m=8[kg].

4.2 Power estimator and Power from DC link

For the value of the Power generated I initially used the average value from the Chopper. Remembering that one dissipates power only when the resistance's switch is closed (otherwise the energy would get stored in the capacitor) one could find the average power from a chopper's cycle:

$$P_{avgDC} \approx \frac{\frac{V_{avg}^2}{R_{chop}} \cdot T_{close} + 0 \cdot T_{open}}{\Delta T} \approx \frac{V_{avg}^2}{R_{chop}} \cdot \frac{T_{close}}{\Delta T} \tag{4.6}$$

where T_{close} was the time of the switch closed, T_{open} was the time of the switch opened and ΔT was the sum of those two times; V_{avg} was the average voltage in the cycle and R_{chop} was the chopper resistance. But this value, representing the electrical power available for the grid, was much smaller than the mechanical power, due to the large losses in my small electrical machine (in laboratory).

So I implemented also a Power Estimator with an approach similar to [14], but adapted for our IG with DELTA connection.

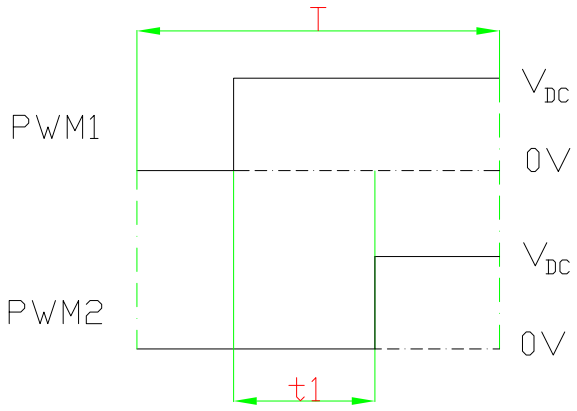


Figure 4.6: *Average phase voltage in one interrupt (IG Delta-connection)*

$$V_{AVG} = \frac{t_1}{T} \cdot V_{DC} \qquad (each\ interrupt) \tag{4.7}$$

$$I_{meas} = \frac{I_1 - I_2}{3} \qquad (each\ interrupt) \tag{4.8}$$

$$P_{tot} = \frac{\sum^N V_{AVG} \cdot I_{meas}}{N°sample} \qquad (every\ period) \tag{4.9}$$

$$I_{rms} = \frac{\sum^N I^2_{meas}}{N°sample} \qquad (every\ period) \tag{4.10}$$

Obtaining the Power

$$P_{elec} = 3 \cdot (P_{tot} - I_{rms} \cdot R_{phase}) \qquad (every\ period) \tag{4.11}$$

However this value was also much lower than the mechanical power, as it was working with a small Induction machine with low efficiency. *For my tests I will use the Mechanical Power obtained from the measure instrument* (but this last explained approach can be used in bigger Induction Generator like the one in the Gaia Wind turbine).

4.3 Correspondence between Simulation and Gaia WT

In the simulations I tried to remain connected to the real Wind Turbine with the values of Torque, Rotational Speed and Power. Therefore rewriting (*eq:*4.3):

$$T_{GAIA} = \gamma_T \cdot T_{lab}; \qquad \Omega_{GAIA} = \frac{\Omega_{lab}}{\gamma_\Omega} \tag{4.12}$$

with the relation:

$$T_{GAIA} \cdot \Omega_{GAIA} = (T_{lab} \cdot \gamma_T) \cdot (\frac{\Omega_{lab}}{\gamma_\Omega}) = K_p \cdot T_{lab} \cdot \Omega_{lab} \tag{4.13}$$

$$K_p = \frac{\gamma_T}{\gamma_\Omega} \approx \frac{223}{1.5} \approx 149 \tag{4.14}$$

and from (*eq:*1.6)

$$P_{GAIA} = K_p \cdot P_{lab} \tag{4.15}$$

with the Rated Power of 11[KW] of Gaia Wind, my rated power in the laboratory was $P_{MAX-lab} = 74[W]$. So for Torque, Rotational Speed and Power the relationships were:

	from laboratory
$T_{GAIA} =$	$223 \cdot T_{lab}$
$\Omega_{GAIA} =$	$\Omega_{lab}/1.5$
$P_{GAIA} =$	$149 \cdot P_{lab}$

Table 4.1: Summarize of relationships

Chapter 5

Development 1: Control of Induction Generator

In this chapter the Visual Basic interface used in the laptop and the "C-program" running in the FCIV-controller will be presented for the side of the Induction Generator. However I will start explaining how I created the sinusoidal 3-phases from Frequency and Modulation Index values.

5.1 Discrete sinusoid from Frequency and Modulation Index

The values of Frequency and Modulation Index were changeable input from the Visual Basic interface. They were sent to the FCIV-controller and they were translated in the sinusoidal 3-phases in the "C-program". In order to did it, I used an array with 2040 elements of a quantized sinusoid with values between the maximum and the minimum available. The Frequency asked was translated in numbers of array's elements used in each signal period (samples), and consequently in the size of the "jump" for change the element after one interrupt. Please keep in mind that the Frequency of PWM (10KHz) is the reciprocal of the Interrupt Time (100μs):

$$samples = \frac{f_{PWM}}{f_{sin}} = \frac{10000}{f_{sin}} \tag{5.1}$$

$$jump = \frac{Total\ elements\ array}{samples} = \frac{2040}{samples} \tag{5.2}$$

Therefore a sinusoid at 50[Hz] had 50 periods in 1[s] each composed by 200 samples of the array (jump=10.2). For example the series of array values (for one phase) could be:

$$v_0 \rightarrow v_{10} \rightarrow v_{20} \rightarrow v_{30} \rightarrow v_{40} \underbrace{\overset{+1}{\rightarrow} v_{51}}_{rest\ added} \rightarrow v_{61} \rightarrow v_{71} \rightarrow ...$$

If the "jump" was not a integer, it is implied that I used its integer value, memorizing the rest and using it, like additional jump, when it went over 1.

After the sinusoid creation the Modulation Index was simply used like value between 0 and 1 ($0 < m_i < 1$) to reduce the amplitude of the sinusoid.

The 3-phases (120°) were created using 3 independent pointers, each after 1/3 of the length of the array, for example:

$$\left[\underbrace{v_0}_{pointer\ 1}\ v_1\ v_2\ \cdots\ v_{679}\ \underbrace{v_{680}}_{pointer\ 2}\ v_{681}\ \cdots\ v_{1359}\ \underbrace{v_{1360}}_{pointer\ 3}\ v_{1361}\ \cdots\ v_{2038}\ v_{2039} \right]$$

5.2 Induction Generator: Feature

With an example figure I want to explain what happened changing the frequency (and m_i using V/Hz control) to the Induction Generator.

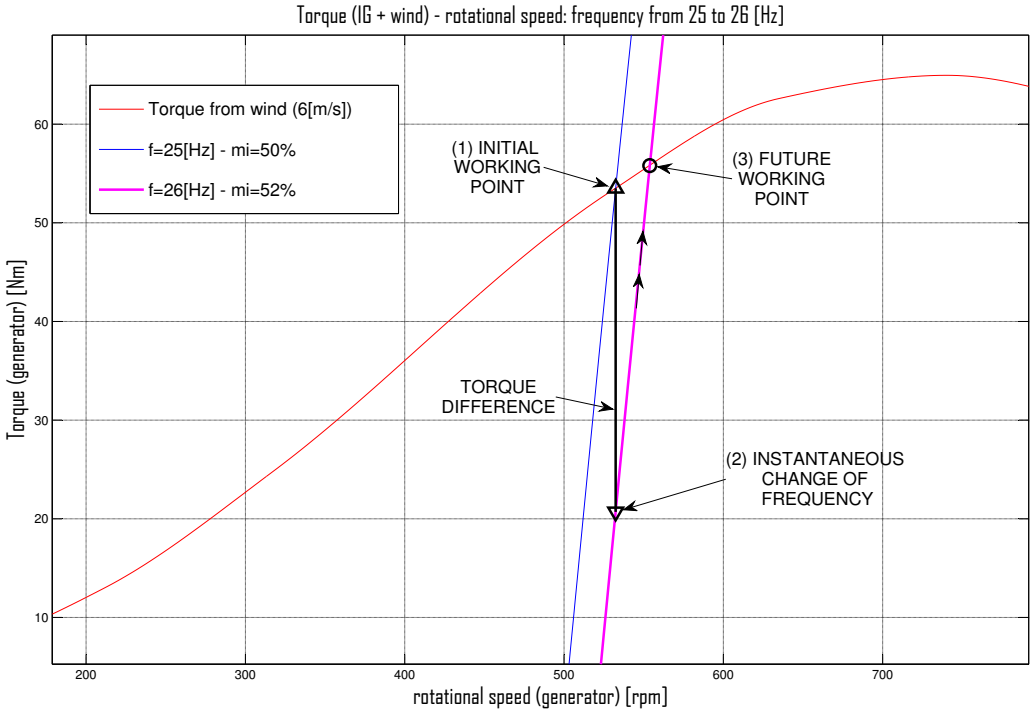


Figure 5.1: *Change of frequency (V/Hz control) for fixed wind speed*

In the example, increasing the frequency in the Induction Generator from 25 to 26 [Hz] (m_i from 50 to 52 [%]) the Rotational Speed did not change instantanely and the generator started to work in the new IG curve (point 2 in fig.). The difference of Torque ($T_{wind} > T_{IG}$) increased the Rotational Speed reaching the new working point where the torques were balanced again (point 3 in fig.). Indeed the change of frequency (& m_i) had to be very small due to the big slope of the IG curve.

5.3 Induction Generator: Visual Basic Interface

The Visual Basic Interface was composed of 3 INPUT (from interface to controller) on the left:

- Frequency [Hz] (for V/Hz control)
- Modulation index [%] (for V/Hz control)

- Change speed [Hz/s] (speed of frequency change from last value to new value)

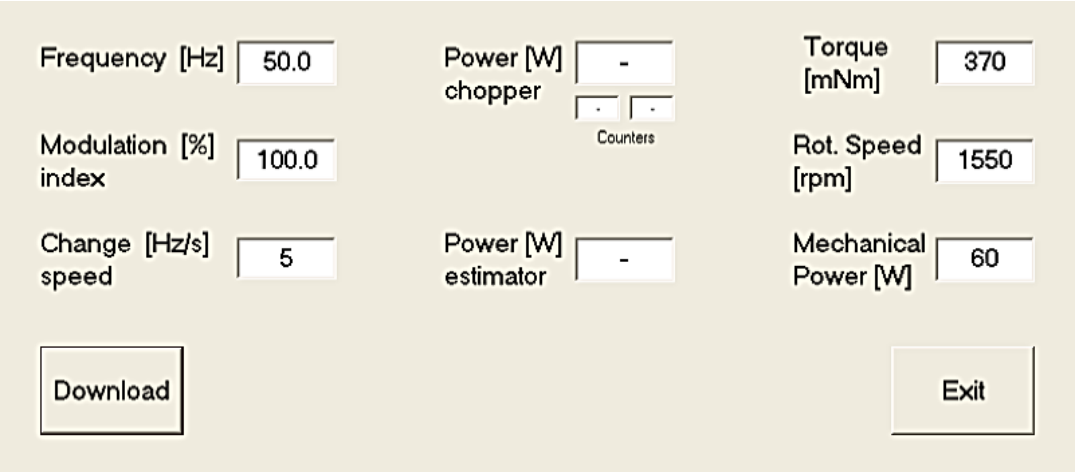


Figure 5.2: *Induction Generator interface*

And 5 OUTPUT (from controller to interface) on the right:

- Power chopper [W] (power calculated from DC-link, with counter charging and decharging)
- Power estimator [W] (power calculated from PWM and current measure)
- Torque [mNm] (torque measure from "Torque and Rotational speed display -UMV 2000-")
- Rotational speed [rpm] (rotational speed measure from UMV 2000)
- Mechanical Power [W] (power calculated from torque and rotational speed)

The "Change speed" also setted the speed of Modulation Index change: from 0[%] to 100[%] in the same time of the frequency change $0[Hz] \rightarrow 50[Hz]$. If I had for example Change speed = 5[Hz/s], in 2[s] I could slip from ($f = 40[Hz]$ & $m_i = 80\%$) to ($f = 50[Hz]$ & $m_i = 100\%$).

5.4 Induction Generator: C programs

The "C-project" for the FCIV-controller was a complex set of files with many purposes: initialize, variable definition, system control, ADC management, interrupt management... All this staff was already created by SPEED Laboratory so I will examine only the realized main file, composed of 600 lines totally presented in Appendix B.

I start presenting a communication example from the many used, where the FCIV-controller sent the values of Torque, Rotational Speed and Mechanical Power to the Visual Basic Interface:

Communication example IG

```
if (RSRx[4] == 32) { /* Power sent to PW */
    RotSpeed_s = RotSpeed_aver; // Rotational Speed
    Torque_s = Torque_aver; // Torque
    PowerMec_s = PowerMec; // Mechanical Power
    Generating_s = Generating; // boolean generating/motoring
    if(Torque_s<0){ // if motoring
           Torque_s = -Torque_s;
           PowerMec_s = -PowerMec_s; }
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RotSpeed_s/256; // Rotational Speed MSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RotSpeed_s%256; // Rotational Speed LSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Torque_s/256; // Torque MSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Torque_s%256; // Torque LSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerMec_s/256; // Mechanical Power MSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerMec_s%256; // Mechanical Power LSBs
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Generating_s; // boolean gener./motor.
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0; }
```

In the example: firstly came to the controller the request of data from the interface (managed by a timer). This request was a communication package with value 32 in the byte 4. The controller checked all the possible requests, and when it identified the correct one (in the code: RSRx[4]==32), it put in the buffer the values that the interface was attending. Usually each variable was sent by 2 bytes, with the value divided by 256 and the rest.

Now I present the creation of the sine output (translated in PWM) in one interrupt, like showed in (*sec:*5.1).

Phase 1: sinusoid creation

```
/*----- Fetch value from Look-Up Table ---- */
/*--- Phase 1 ---*/
phase1_Point += jump_Int; // increase of jump
phase1_Rest += jump_Rest; // increase of rest
if (phase1_Rest >= 1){ // rest management
    phase1_Rest -= 1;
    phase1_Point += 1; }
if (phase1_Point > (Tot_elem-1)){ // circular array management
    phase1_Point = phase1_Point - Tot_elem; }
IRt1 = IrefPhase_H1875_L2040[phase1_Point]; // value from array
xSin1 = IRt1*Mod_act/Mod_MAX; // use of modulation index
DutyC1 = max + xSin1; // PWM translation
```

The Frequency and the Modulation Index were changed slowly respecting the parameter “Change speed” chosen. So for every interrupt the Frequency and the Modulation Index were incremented (or decremented) until reach the desired value, with steps corresponding to the interrupt time (100μs).

Slow change of Frequency and Modulation Index

```
// Slow change Frequency
if(changing_F!=0){
   // additional step of frequency
   Freq_act += (coef_Freq*changing_F*Hz_s*timeInter/1000);
   // reached value of demanded ‘‘frequency’’
   if((changing_F==1 && Freq_act>FREQ)||(changing_F==-1 && Freq_act<FREQ) ){
      Freq_act=FREQ;
      changing_F=0; }
   // elements of the array used
   elements = 1000/(timeInter*(Freq_act/coef_Freq));
```

```
   jump_Int = (int)(Tot_elem/elements); // new jump
   jump_Rest = (Tot_elem/elements)-jump_Int; // new rest
}
// Slow change Modulation Index
if(changing_M!=0){
   // additional step of modulation index
   Mod_act += (changing_M*(Mod_MAX/Mod_s)*timeInter/1000);
   // reached value of demanded ‘‘modulation index’’
   if((changing_M==1 && Mod_act>MOD_IND)||(changing_M==-1 && Mod_act<MOD_IND) ){
      Mod_act=MOD_IND;
      changing_M=0; }
}
```

In the code “elements” corresponds to “samples” explained in (*eq:*5.1) and (*eq:*5.2).

Chapter 6

Development 2: Wind Emulator

In this chapter the Visual Basic interface used in the laptop and the "C-program" running in the FCIV-controller will be presented for the side of the Wind Emulator. An existing general program (for the control of PMAC, DC and Switched Reluctance motors) was used. Therefore only the 2 modified panels of the Visual Basic Interface, and the developed parts of the "C-program" will be presented.

6.1 Wind Emulator: Feature

The curve $C_t - \lambda$ showed in *Figure 1.6*, necessary for the torque control, was implemented through a look-up table. After the creation of a reasonable $C_t - \lambda$ curve from the GAIA Wind data, I saved these values in an array with the opportune discretization: one C_t value for every 0.1 λ (between the values 0 and 20).

Similarly to (*sec:*5.2), I try to explain what happened changing the Wind Speed with an example (*Figure 6.1*), with the Wind Speed decreasing from 5 to 4 [m/s]. In the Induction Generator nothing changed instantly, however the Torque demand from IG was bigger than the Torque available from the wind ($T_{wind} < T_{IG}$): the Rotational Speed decreased to reach the new working point where the torques were balanced again (point 2 in fig.).

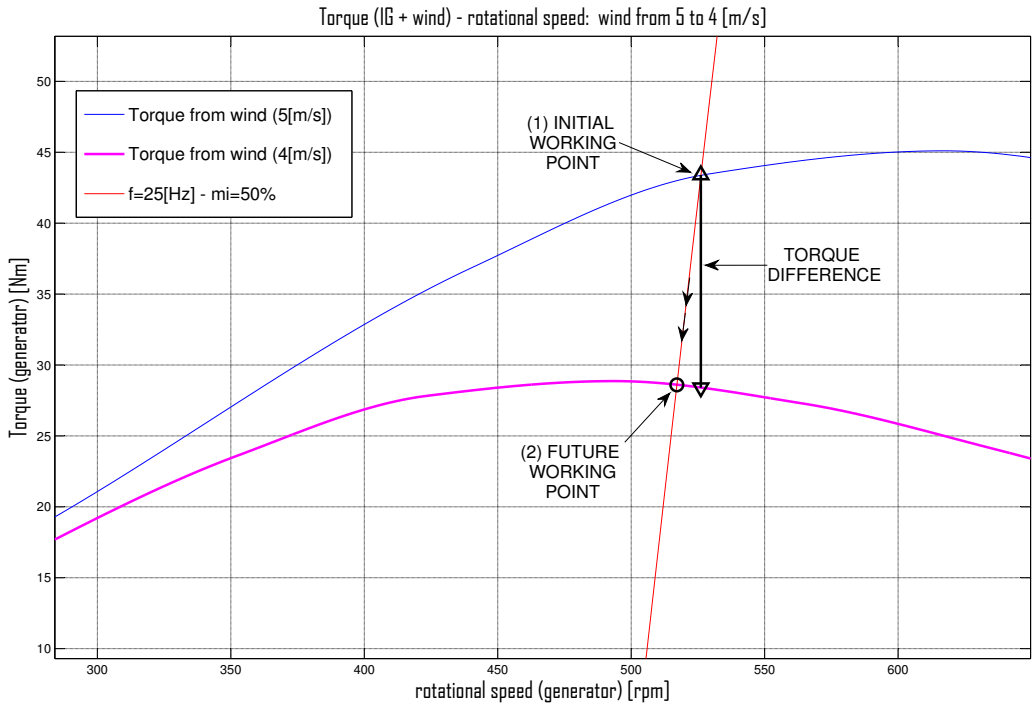


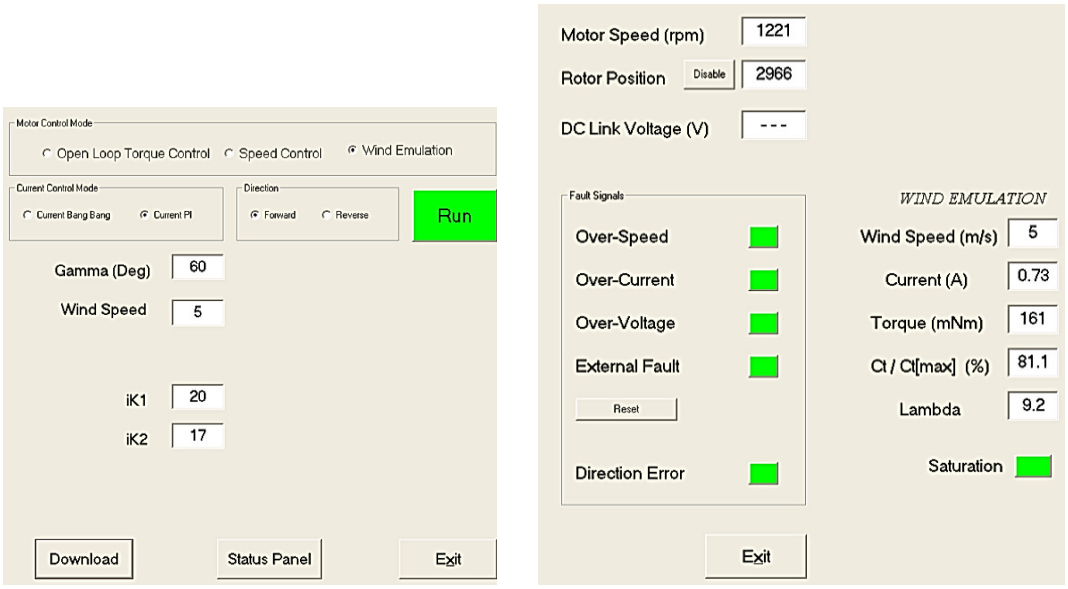
Figure 6.1: *Change of Wind Speed for IG with fixed parameters*

6.2 Wind Emulator: Visual Basic Interface

The Wind Emulator was interfaced with 2 Visual Basic panel.

The **PMAC motor interface**, *Figure 6.2(a)*, was the main panel where in the area *Motor Control Mode* was added the flag *Wind Emulation*. Choosing this flag, the main panel appeared like showed, where the other two possible inputs "Current reference" and "Speed reference" disappeared, and only the "Wind Speed" input remained.

Pushing the *Status Panel* button in the "PMAC motor interface" the **Status panel interface** appeared as showed in *Figure 6.2(b)*. This panel was already implemented with the Motor Speed and a useful list of Fault Signals (Over speed-current-voltage and external Fault); in the right I integrated the list of outputs necessary in the Wind Emulation control.



(a) PMAC Motor Interface (b) Status Panel Interface

Figure 6.2:

The list of outputs was composed by:

1. Current [A] (translated from torque for current control)
2. Torque [mNm]
3. Ct/Ct(MAX) [%] (percentage of torque coefficient)
4. Lambda [·] (tip speed ratio)
5. current Saturation [on/off]

6.3 Wind Emulator: C programs

As explained for the Induction Generator side, also in the Wind Emulator side the "C-project" for the FCIV-controller was a complex set of files with many purposes: initialize, variable definition, system control, ADC management, interrupt management... In addiction the main "C-program" was finally composed of more than 2000 lines of code, for the control of PMAC, DC and Switched Reluctance motors. Only the developed parts in the main program will be presented.

The communication with the Visual Basic Interface was similar to how described in the IG case:

Communication example WE

```
if (RSRx[4] == 23)
  {
  /* Daniele_2 */
  WindSp_s = WindSp; // Wind Speed 0-->25 [10x] 0-->250
  Current_s = CurrentRef; // Current 1A-->409
  Torque_s = (int)Torque ; // Torque [mNm] 0-->1000
  Ct_CtMAX_s = (int)(1000*Ct_LI/CtMAX); // percentage torque coefficient [10x] 0-->1000
  Lambda_s = (int)(10*Lambda); // tip speed ratio 0-->20 [10x] 0-->200
  Saturated_Curr_s = Saturated_Curr; // boolean current saturation

  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Saturated_Curr_s; // boolean current saturation
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = RSRx[1]; // ---
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = WindSp_s/256; // Wind Speed MSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = WindSp_s%256; // Wind Speed LSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Current_s/256; // Current MSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Current_s%256; // Current LSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Torque_s/256; // Torque MSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Torque_s%256; // Torque LSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Ct_CtMAX_s/256; // percentage torque coefficient MSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Ct_CtMAX_s%256; // percentage torque coefficient LSBs
  while (!(SciaRegs.SCICTL2.all & 0x80));
  SciaRegs.SCITXBUF = Lambda_s; // tip speed ratio
  }
```

In the example: firstly the request of data from the interface (managed by a timer) came to the controller. This request was a communication package with value 23 in the byte 4. The controller checked all the possible requests, and when it identified the correct one (in the code: RSRx[4]==23), it put the values that the interface is attending in the buffer. Usually each variable was sent by 2 bytes, with the value divided by 256 and the rest. The variables needed with more

accuracy (for example 0.1) were sent after a multiplication for a constant (ex. 10) and converted again in the Visual Basic panel.

In the Wind Emulator, the program calculated the PMAC motor's torque (that represents the torque in the turbine) from the Wind Speed input and the Rotational Speed. At last, for every input, the controller translated this torque in current, using a constant found in a laboratory test, and worked like a current control.

Torque (current) control

```
//-------------------------------------------------------------
if(LpMd==3) {
  rpmCount +=1;
  if(rpmCount>=60) { // 3ms/6ms
      rpmCount=0;
           //----Daniele variable wind_data
      // WindCount +=1;
      // if(WindCount >= 20) { // 60ms/120ms
      // WindCount=0;
      // V_wind = wind_data_1[WindPointer]; // download wind speed from array
      // if(WindPointer >= (DataWind-1)) WindPointer=0; // restart simulation
      // else WindPointer += 1;
      // }
      V_wind = (double)WindSp/10;
      if(V_wind<0.1) { //wind speed used like divisor in the lambda equation
           Torque=0;
           Lambda=20; // high saturation
           LambdaPointer=(int)(Lambda*DiscrLambda);
           Ct_LI = Ct_lambda_curve_INTdecim[LambdaPointer]; // torque coefficent
           CurrentRef=(TranslCurrent*Torque/K_Torq); // current amplitude (NO offset) 409=1[A]
           }
      else {
           RPM=(int)((625.0/128.0)*deltaPos); // rotational speed[encoder incremental]-->[rpm]
           Lambda=2*piValue*Radius*RPM/(1.5*60*nGearbox*V_wind); // tip speed ratio
           if(Lambda<0) Lambda=0; // low saturation
           if(Lambda>20) Lambda=20; // high saturation
           LambdaPointer=(int)(Lambda*DiscrLambda);
            // torque coefficent from array (rappresent the curve Ct-lambda)
           Ct_LI = Ct_lambda_curve_INTdecim[LambdaPointer];
           Torque=(rho*piValue*(Radius*Radius*Radius)*(Ct_LI/1000)*(V_wind*V_wind))/(2*
                  TranslTorque*nGearbox); // torque [mNm] traslated (156Nm-->0.7Nm)
            // current amplitude 409=1[A]
           CurrentRef=(TranslCurrent*Torque/K_Torq)+(TranslCurrent*CurrentOffset);
           }
      if(CurrentRef>(2.2*TranslCurrent)) { // saturation 2.2 [A]
           Saturated_Curr=1;
```

```
          CurrentRef=(2.2*TranslCurrent);
          }
     else {
          Saturated_Curr=0;
          }
     }
  }
else{
  WindSp=0;
  Torque=0;
  Lambda=0;
  Ct_LI=0;
  }
//-------------------------------------------------------
```

Chapter 7

Laboratory Tests

In this chapter I will briefly present all the principal laboratory's tests. They are in steady states, necessary to understand future dynamic implementation.

7.1 Pre-test

Using the CURRENT CONTROL in the VB interface (on the WE side), I made a test to determine the constant that translate the desired torque in current reference (K_{Torque}).

f=50Hz , mi=100%						
rot. speed [rpm]	1475	1500	1536	1558	1580	*1590*
Torque [mNm]	-77	40	265	450	640	*710*
Curr. PMAC [A]	0,1	0,4	1	1,5	2	*2,2*

f=25Hz , mi=50%						
rot. speed [rpm]	735	760	790	820	830	*840*
Torque [mNm]	-42	65	280	450	630	*710*
Curr. PMAC [A]	0,1	0,4	1	1,5	2	*2,2*

Figure 7.1: *Test: Current-Torque*

So I took a table with the Current reference Input (from 0.1 to 2.2 [A]) and the Torque [mNm] measured. I made the test two time: for Induction Generator with ($f = 50[Hz]$ & $m_i = 100\%$) and ($f = 25[Hz]$ & $m_i = 50\%$), obtaining the relation:

$$I = I_{offset} + K_{Torque} \cdot T = 0.3 + 375 \cdot T \tag{7.1}$$

where (I) is the current, and (T) the torque. In the table I also took the values of Rotational Speed: analizing only Torque & Rotational Speed togheter I obtained the first and the last curves showed in *Figure 2.7* at pag. 20.

7.2 Test: Wind speed change

Now the first important test is presented, where for fixed Frequency and Modulation Index in the IG, I checked what happen for the change of the Wind Speed. The test was repeated three times, in three different lines, for low ($f = 25[Hz]$ & $m_i = 50\%$), medium ($f = 38[Hz]$ & $m_i = 76\%$) and high ($f = 50[Hz]$ & $m_i = 100\%$) values of Frequency and Modulation Index in the IG.

	laboratory							
	v [m/s]	I [A]	Ct [%]	lambda	T [mNm]	wm [rpm]	Pmec [W]	P_GAIA [W]
	0	0	0,1	20	-80	720	-6	-899
	3,5	0,54	92,7	8,2	110	770	9	1322
	5,5	0,85	86,9	5,3	225	780	18	2738
f = 25 Hz	7	0,98	65,2	4,2	270	790	22	3328
mi = 50%	8,5	1,03	48,8	3,5	280	800	23	3495
	10	1,1	38,1	3	300	805	25	3768
	15	1,26	19,9	2	365	810	31	4613
	25	1,64	10,5	1,2	500	820	43	6397
	0	0	0,1	20	-90	1115	-11	-1566
	3,5	0,4	40,2	12,3	60	1145	7	1072
f = 38 Hz	5,5	0,9	94,4	8	240	1175	30	4400
mi = 76%	7	1,32	98,3	6,4	395	1195	49	7365
	8,5	1,61	86,9	5,3	500	1205	63	9401
	10	1,8	71,9	4,5	570	1215	73	10806
	15	2,11	38,1	3	690	1225	***89***	***13189***
	25	SATUR.						
	0	0	0,1	20	-105	1470	-16	-2408
	3,5	0,32	8,6	16,1	15	1500	2	351
f = 50 Hz	5,5	0,72	65	10,4	170	1525	27	4045
mi = 100%	7	1,24	91,7	8,3	370	1550	60	8948
	8,5	1,85	99,8	7	600	1580	***99***	***14792***
	10	SATUR.						

Figure 7.2: *Test: wind speed change for fixed frequency and modulation index*

In the first column there is the wind speed (v[m/s]), in the following three columns there are the current (I[A]), the percentage value of the torque coefficient (Ct[%]) and the tip speed ratio (lambda: λ). After there are the Torque (T[mNm]), the Rotational Speed Ω_m (wm[rpm]) and the laboratory's Mechanical Power (Pmec[W]) calculated from the previous two. In the last column the Laboratory's power is translated in the GAIA Wind's power how explained in (*sec:*4.3).

Please remember that the GAIA Wind rated power was 11[Kw], associated to 74[W] in laboratory. Powers bigger than these are showed to understand the WT's behavior, but were not real in the work range. Moreover (and over the rated power threshold) in the Wind Emulator's controller there was a current limitation of $I_{MAX} = 2.2[A]$, associated to the torque $T_{MAX} = 700[mNm]$: this was the maximum value of the torque in the generator shaft, for the maximum wind speed $v_{MAX} = 25[m/s]$ at the rotational speed of the blade 56[rpm]. In case of current saturation only the flag SATUR. is showed and no bigger wind speeds were analyzed.

7.3 Test: Frequency and Modulation Index change

In this second test I present in a similar table an opposite test *(Figure 7.3)*: for fixed Wind Speed I changed Frequency and Modulation Index from the minimum value ($f = 25[Hz]$ & $m_i = 50\%$) to the maximum value ($f = 50[Hz]$ & $m_i = 100\%$) using the V/Hz control (*sec:*2.3) with step of $\Delta f = 5[Hz]$ & $\Delta m_i = 10\%$. The test was repeated four times, in four different lines, for low ($v = 3.5[m/s]$), medium ($v = 5.5[m/s]$) and high ($v = 7[m/s]$) values of Wind Speed in the range MPPT; the fourth line is for a Wind Speed over this range ($v = 9[m/s]$), where was not possible to apply the Maximum Power Point Tracking.

In the first column there are the Frequency and Modulation Index ($f[Hz] - m_i[\%]$) from the V/Hz control, and the rest of the table is the same explained for the previous test: in the following three columns there are the current (I[A]), the

percentage value of the torque coefficient (Ct[%]) and the tip speed ratio (lambda: λ).

		laboratory						
	f [Hz] - mi [%]	I [A]	Ct [%]	lambda	T [mNm]	wm [rpm]	Pmec [W]	P_GAIA [W]
MPPT-low	25 - 50	0,54	92,7	8,2	125	775	10,1	1512
	30 - 60	0,49	73,5	9,8	95	910	9,1	1349
v = 3.5 [m/s]	35 - 70	0,43	52,2	11,3	75	1055	8,3	1235
	40 - 80	0,38	33,8	12,9	55	1200	6,9	1030
	45 - 90	0,34	19,3	14,5	35	1350	4,9	737
	50 - 100	0,32	8,6	16,1	20	1500	3,1	468
	25 - 50	0,87	88,7	5,4	235	780	19,2	2860
MPPT-medium	30 - 60	0,93	98,3	6,4	255	940	25,1	3740
v = 5.5 [m/s]	35 - 70	0,93	98	7,5	260	1085	29,5	4402
	40 - 80	0,87	89,5	8,5	235	1235	30,4	4528
	45 - 90	0,8	78,6	9,4	205	1380	29,6	4414
	50 - 100	0,72	65	10,4	170	1520	27,1	4032
	25 - 50	0,98	65,2	4,2	280	790	23,2	3451
MPPT-high	30 - 60	1,16	83,4	5,1	345	955	34,5	5141
v = 7 [m/s]	35 - 70	1,3	96,2	5,9	390	1110	45,3	6755
	40 - 80	1,35	99,4	6,7	400	1250	52,4	7802
	45 - 90	1,32	98	7,5	400	1405	58,9	8769
	50 - 100	1,24	91,7	8,3	370	1550	60,1	8948
	25 - 50	1,04	44,3	3,3	300	800	25,1	3745
OVER-mppt	30 - 60	1,34	60,5	4	405	955	40,5	6035
v = 9 [m/s]	35 - 70	1,56	74	4,6	485	1115	56,6	8438
	40 - 80	1,78	86,9	5,3	565	1275	**75,4**	**11240**
	45 - 90	1,95	96,2	6	630	1430	**94,3**	**14057**
	50 - 100	1,99	99,1	6,6	655	1585	**108,7**	**16199**

Figure 7.3: *Test: frequency and modulation index change for fixed wind speed*

After there are the Torque (T[mNm]), the Rotational Speed Ω_m (wm[rpm]) and the laboratory's Mechanical Power (Pmec[W]) calculated from the previous two. In the last column the Laboratory's power is translated in the GAIA Wind's power as explained in (*sec:*4.3).

We can see that for slow Wind Speed the maximum Power was obtained for the lower frequency (and m_i), and for high Wind Speed the maximum Power was for higher frequency. Resuming it in a figure (for the MPPT range):

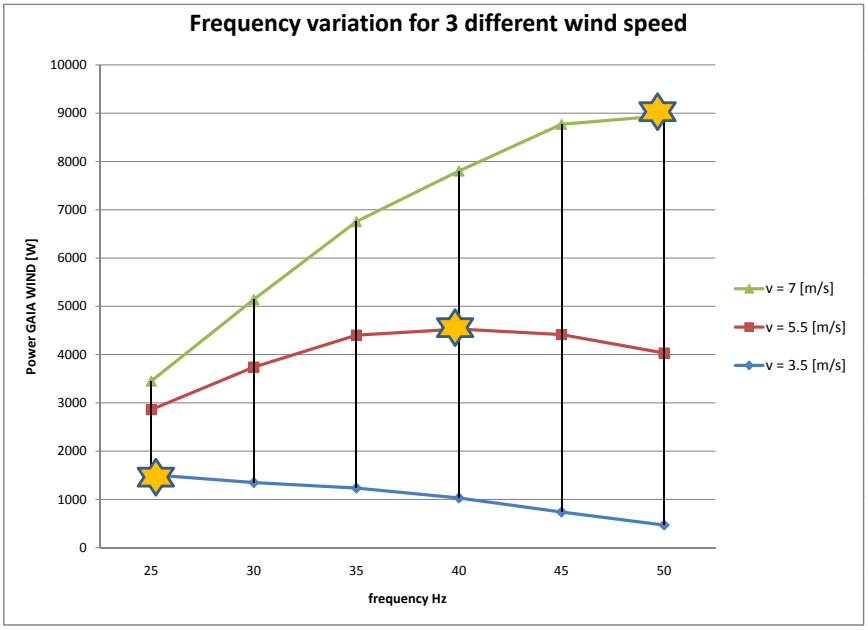


Figure 7.4: *Test: MPPT*

7.4 Test: Fixed-Speed of GAIA Wind

This test is similar to the test presented in (*sec:*7.2) where we checked what happen with changing speed of the wind. But this time we emulated what happen in the real GAIA Wind turbine with Gear-ratio $18 \div 1$ and fixed Frequency (50[Hz]) and Modulation Index (100[%]) for the Induction Generator.

	laboratory						
	v [m/s]	I [A]	Ct [%]	lambda	T [mNm]	wm [rpm]	Pmec [W]	P_GAIA [W]
	0	0	0,1	20	-100	1470	-15,4	-1539
	3,5	0,45	59,1	10,8	70	1505	11,0	1103
	5,5	0,94	99,8	7	250	1535	40,2	4019
50 Hz - 100 %	7	1,23	90,3	5,5	360	1550	58,4	5843
GEARBOX **18-1**	8,5	1,41	74	4,6	425	1558	69,3	6934
FIXED SPEED	10	1,52	58,2	3,9	470	1562	76,9	7688
	15	1,73	30,2	2,6	550	1572	90,5	9054
	25	1,99	13,6	1,5	650	1585	107,9	10789
					! NOT compare with other tests !			

Figure 7.5: *Test: Fixed-Speed of GAIA Wind*

This table is necessary to compare the response of the Power (P_GAIA) in Variable and Fixed speed functioning for different Wind Speed. However please make ATTENCTION that we can not compare the laboratory's values, because they use different ranges and multiplication constants (in Fixed Speed: $\gamma_T = 150, K_p = 100$ and $P_{MAX-lab} = 110[W]$) due to the different gear-ratio.

In the first column there is the wind speed (v[m/s]), in the following three columns there are the current (I[A]), the percentage value of the torque coefficient (Ct[%]), and the tip speed ratio (lambda: λ). After there are the Torque (T[mNm]), the Rotational Speed Ω_m (wm[rpm]), and the laboratory's Mechanical Power (Pmec[W]) calculated from the previous two. In the last column the Laboratory's power is translated in the GAIA Wind's power with the different constant showed above.

7.5 Comparison Fixed-Variable Speed

From all the tests seen above I obtained this resuming table:

v [m/s]	P_Fixed [W]	P_Variable [W]
0	-1539	-899
3,5	1103	1512
5,5	4019	4528
7	5843	8948
8,5	6934	11000
10	7688	11000
15	9054	11000
25	10789	11000

Figure 7.6: *Result: comparison Fixed-Variable Speed*

where the powers showed are both P_GAIA values from previous tests:

- P_Fixed[W] is the Power from the Fixed-Speed test in *Figure 7.5*
- P_Variable[W] is the Power from the Variable-Speed tests in *Figure 7.2* and *Figure 7.3*

Using the same Wind Speed, 4-5-6-7[m/s], used in the GAIA Wind data-sheet, *Figure 1.10* at pag. 11, I calculated the corresponding power through a weighted average and the AEP (Annual Energy Production) using an estimated 90% of turbine's activity.

FIXED sp.	Pavg [W]	(AEP) [kWh]
4m/s	1832	14444
5m/s	3290	25937
6m/s	4627	36478
7m/s	5843	46069

(a) Fixed-speed emulation

(AEP)
16,220 kWh
27,502 kWh
37,959 kWh
46,527 kWh

(b) Datasheet

VARIABLE sp.	Pavg [W]	(AEP) [kWh]
4m/s	2266	17865
5m/s	3774	29754
6m/s	6001	47315
7m/s	8948	70546

(c) Variable-speed emulation

Figure 7.7:

We can see the correspondence between the laboratory's WE working in Fixed Speed (a) and the data from Datasheet (b). So comparing the Fixed (a) and the Variable-speed functioning (c) I obtained for the AEP values:

- $4[m/s] \quad \Rightarrow \quad +3400[kWh] \quad \Rightarrow \quad +23\%$
- $5[m/s] \quad \Rightarrow \quad +3800[kWh] \quad \Rightarrow \quad +15\%$
- $6[m/s] \quad \Rightarrow \quad +10800[kWh] \quad \Rightarrow \quad +30\%$
- $7[m/s] \quad \Rightarrow \quad +24450[kWh] \quad \Rightarrow \quad +53\%$

This results shows that theoretically the Variable-Speed functioning permits a huge increase of Power captured (in the steady state analysis). The gain is low for $v \approx 5[m/s]$, because the Fixed-Speed functioning works near the optimum point, however the gain increases significantly for the other Wind Speeds, above all for high values where the optimum power increase quickly.

This is a good starting point for the future dynamic analysis, where the management of the Rotational Speed will certainly reduce the gain showed.

Conclusions

In this project work the equations of a Wind Turbine and the behaviour of an Induction Generator were studied. Starting from the Fixed-Speed functioning of the GAIA Wind turbine 11[KW] a strategy for the Variable-Speed functioning was implemented, showing a huge increment of available power.

Developing a Matlab Simulation, the gain of Variable-Speed functioning in the MPPT (maximum power point tracking) was found being really available. However, for power near the Rated value, some problems might occour, due to the energy stored in the blade's Inertia associated with the limitation of a small wind turbine, having NOT managing of the pitch regulation. Such problems were described and will be object of future analysis.

At last a hardware test facility was developed, composed of an Induction Generator, PMAC Wind Emulator, and relative controllers. With this Rig we did many steady state tests, highlighting the relationships between the results obtained in laboratory and the results obtainable in the real wind turbine.

Appendix A

Glossary

A.1 Definitions

AEP = annual energy production

IG = induction generator

IGBT = insulated gate bipolar transistor

MPPT = maximum power point tracking

PMAC = permanent magnet AC

VB = visual basic

WE = wind emulator

WT = wind turbine

P = power

T = torque

Ω_m = rotational speed generator shaft (mechanical)

Ω_o = synchronous speed generator (mechanical)

ω = rotational speed blades shaft

v = wind speed

C_t = torque coefficient

C_p = power coefficient

λ = tip speed ratio (lambda)

R = blade radius

ρ = air density

J = inertia

n_r = gear ratio of gearbox

f = frequency

m_i = modulation index

s = slip

p = pole pairs

γ_t = torque constant between laboratory and GAIA wind

γ_Ω = rotational speed constant between laboratory and GAIA wind

K_p = power constant between laboratory and GAIA wind

“opt”=optimum

Appendix B

program -C- for the IG controller

It is presented the total -C- program writed for the control of the Induction Generator.

B.1 program -C- for Induction Generator

```
/**********************************************************************
* File: lab12_main.c -- Daniele
* History: 09/18/02 - original (based on DSP28 header files v0.58)
**********************************************************************/
#include "DSP28_Device.h" // Peripheral address definitions
#include "wave_H1875_L2040.h" //----Array

int CalCount2;
int CalCount;
int CalCount3;
long int IRt1;
int CalRxF;
int RSRx[11];
int Iph1,Iph2,Iph3,Iph4;
int CalRP;
int Iph1_1,Iph1_2,Iph1_3;
long int Iph1_L;
int CalDigIn1,CalDigIn2,CalDigIn;
int AI1,AI2,AI3,AI4,ASCB;
int Command, CurrentRef;
int TxErr1;
//----Daniele
double timeInter;
int Tot_elem;
int max;
```

```
int FREQ;
int MOD_IND;
double elements;
int jump_Int;
double jump_Rest;
int xSin1,xSin2,xSin3;
int phase1_Point,phase2_Point,phase3_Point;
double phase1_Rest,phase2_Rest,phase3_Rest;
int DutyC1,DutyC2,DutyC3;
//--------------------------------
double Freq_act;
int changing_F;
int Hz_s;
int coef_Freq;
double Mod_act;
int changing_M;
int Mod_s;
int Mod_MAX;
//--------------------------------
int DanCount;
int Thr_Up;
int Thr_Down;
int Thr_Gen;
int Direct_Thr;
//--------------------------------
int Power;
int Power_s;
int Resist;
int VoltagePow;
int CountPowTot;
int CountPowGen;
//--------------------------------
int CountPowTot_s1;
int CountPowTot_s2;
int CountPowGen_s1;
int CountPowGen_s2;
//--------------------------------
double V_av1,V_av2,V_av3;
int I_meas1,I_meas2,I_meas3;
double PowerAux1;
int CountPowEst;
int Nsample;
int PowerEstTot;
int PowerEstGener;
int PowerEstTot_s;
int PowerEstGener_s;
int ResIG;
//--------------------------------
```

```
int RotSpeed;
int Torque;
int RotSpeed_aver;
int Torque_aver;
double PowerMec;
int Generating;
int RotSpeed_s;
int Torque_s;
int PowerMec_s;
int Generating_s;
//--------------------------------

/* Skeleton Code Definitions */
int AOCon1, AOCon2, CalCount2A, CalCount2B, CalCount2C;
/* Definitions for PC Comms */
#define cmdC1 100 // Parameter 1 Command ID
#define cmdC2 101 // Parameter 2 Command ID
#define cmdC3 102 // Parameter 3 Command ID
/* Definitions for Serial DAC Control */
#define DAC_CHA 0x0000;
#define DAC_CHB 0x4000;
#define DAC_CHC 0x8000;
#define DAC_CHD 0xC000;



/************************************************************************
* Function: main()
* Description: Main function.
************************************************************************/
void main(void)
{
// Initialization
  InitSysCtrl(); // Initialize the CPU (FILE: SysCtrl.c)
  InitGpio();   // Initialize the shared GPIO pins (FILE: Gpio.c)
  InitPieVectTable(); // Initialize the PIE Vectors (FILE: PieVect.c)
  InitPieCtrl(); // Enable the PIE (FILE: PieCtrl.c)
  InitAdc(); // Initialize the ADC (FILE: Adc.c)
  InitEv();  // Initialize the Event Manager (FILE: Ev.c)
  InitSCI(); // Initialise the SCI Interface CC 28/10/03
  InitSPI(); // Initialise the SPI Interface CC 5/11/03

/*** Enable the Watchdog interrupt ***/
  PieCtrlRegs.PIEIER1.bit.INTx8 = 1; // Enable WAKEINT (LPM/WD) in PIE group #1
  PieCtrlRegs.PIEIER5.bit.INTx1 = 1; // Enable T4 Period Interrupt
  IER |= 0x0011;      // Enable INT1 & INT5 in IER to enable PIE group 1

/*** Enable global interrupts ***/
  asm(" CLRC INTM"); // Enable global interrupts
```

```
/*** Reset Variables ****/
  timeInter=0.1; //interrupt time: 0.1[ms]=100[us]
  Tot_elem=2040;
  max=1875; //half value PWM
  //--------------------------------
  changing_F=0;
  Hz_s=5; //change Freq_act[Hz]/[s]
  coef_Freq=10; //[10x] Freq_act=513 -> 51.3[Hz]
  Freq_act=10*coef_Freq; //10.0[Hz]
  changing_M=0;
  Mod_s=50/Hz_s; //change [s] 0->MAX, depend to Hz_s
  Mod_MAX=1000;
  Mod_act=0*Mod_MAX;
  //--------------------------------
  FREQ = 10*coef_Freq; //START 10[Hz]--
  MOD_IND = 0* Mod_MAX; //START 0[V]--
  phase1_Point = 0;
  phase1_Rest = 0;
  phase2_Point = Tot_elem/3;
  phase2_Rest = 0;
  phase3_Point = Tot_elem*2/3;
  phase3_Rest = 0;
  elements = 1000/(timeInter*(Freq_act/coef_Freq));
  jump_Int = (int)(Tot_elem/elements);
  jump_Rest = (Tot_elem/elements)-jump_Int;
  //--------------------------------
  DanCount=0;
  Thr_Up=3968; // 225[V]
  Thr_Down=3840; // 210[V]
  Thr_Gen=3797; // 205[V]
  Direct_Thr=1; // boolean 0(/1)->go down(/up)
  GpioDataRegs.GPFCLEAR.bit.GPIOF13 = 1; //opened switch
  //--------------------------------
  Power=0; //measured Power
  Resist=220; //chopper resistence=220 ohm
  VoltagePow=218; //medium voltage for Power measure (thrUp+thrDOWN)/2
  CountPowTot=0; //total counter
  CountPowGen=0; //generated power counter
  //--------------------------------
  CountPowTot_s1=0;
  CountPowGen_s1=0;
  //--------------------------------
  V_av1=0;
  V_av2=0;
  V_av3=0;
  I_meas1=0;
  I_meas2=0;
```

```
  I_meas3=0;
  PowerAux1=0;
  CountPowEst=0;
  Nsample = (int)elements;
  PowerEstTot=0;
  PowerEstGener=0;
  ResIG=53;
  //--------------------------------
  RotSpeed=0;
  Torque=0;
  RotSpeed_aver=0;
  Torque_aver=0;
  PowerMec=0;
  Generating=0;
  //--------------------------------

  CalCount = 0;
  CalCount2 = 0;
  CalCount3 = 0;
  AOCon1 = 0;
  AOCon2 = 0;
  TxErr1 = 0;
  EvaRegs.ACTRA.all = 0x0666;   // Enable PWM Outputs

  GpioDataRegs.GPACLEAR.bit.GPIOA15 = 1; /* DAC RESET Pulse Low */
  DelayUs(10);
  GpioDataRegs.GPASET.bit.GPIOA15 = 1;

/*** Main Loop ***/
 while(1) // Dummy loop. Wait for an interrupt.
 {
  while (!(SciaRegs.SCIRXST.all & 0x40)); /* NEW APPROACH 29/1/04 */
  {
  RSRx[0] = (SciaRegs.SCIRXBUF.all)&0x00FF;
  /**** 11 Byte Select Test Parameter Interface ****/
  if (RSRx[0] == 11)
   {
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[1] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[2] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[3] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[4] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[5] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   Command = RSRx[5];
```

```
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[6] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[7] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[8] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[9] = (SciaRegs.SCIRXBUF.all)&0x00FF;
   while (!(SciaRegs.SCIRXST.all & 0x40));
   RSRx[10] = (SciaRegs.SCIRXBUF.all)&0x00FF;

   if (RSRx[4] == 18)
   { /* Echo command */
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[0];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[1];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[2];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[3];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[4];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[5];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[6];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[7];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[8];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[9];
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RSRx[10];
    }
//-------------------------------------------------------
   else if (RSRx[4] == 31)
   { /* Power sent to PW */
    Power_s = Power;
    CountPowTot_s2 = CountPowTot_s1;
    CountPowGen_s2 = CountPowGen_s1;
    PowerEstTot_s = PowerEstTot;
    PowerEstGener_s = PowerEstGener;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Power_s/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Power_s%256;
```

```
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = CountPowTot_s2/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = CountPowTot_s2%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = CountPowGen_s2/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = CountPowGen_s2%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerEstTot_s/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerEstTot_s%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerEstGener_s;
   }
//-------------------------------------------------------
 else if (RSRx[4] == 32)
   { /* Power sent to PW */
    RotSpeed_s = RotSpeed_aver;
    Torque_s = Torque_aver;
    PowerMec_s = PowerMec;
         Generating_s = Generating;
    if(Torque_s<0){
           Torque_s = -Torque_s;
           PowerMec_s = -PowerMec_s;
         }
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RotSpeed_s/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = RotSpeed_s%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Torque_s/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Torque_s%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerMec_s/256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = PowerMec_s%256;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = Generating_s;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
    while (!(SciaRegs.SCICTL2.all & 0x80));
    SciaRegs.SCITXBUF = 0;
```

```
   while (!(SciaRegs.SCICTL2.all & 0x80));
   SciaRegs.SCITXBUF = 0;
   while (!(SciaRegs.SCICTL2.all & 0x80));
   SciaRegs.SCITXBUF = 0;
  }
//-------------------------------------------------------
 }
}
if (Command != 0)
  {
  /*******************************************/
  /* Check PC Command Paramter and store value */
  /*******************************************/
       switch (Command)
       {
       case cmdC1: // Frequency
         FREQ = RSRx[6]*256 + RSRx[7];
          if(FREQ>Freq_act)
             changing_F=1;
          if(FREQ<Freq_act)
             changing_F=-1;
          if(FREQ==Freq_act)
             changing_F=0;
       break;
       //-------------------------
       case cmdC2: // modulation index
         MOD_IND = RSRx[6]*256 + RSRx[7];
          if(MOD_IND>Mod_MAX)
             MOD_IND=Mod_MAX;
          if(MOD_IND>Mod_act)
             changing_M=1;
          if(MOD_IND<Mod_act)
             changing_M=-1;
          if(MOD_IND==Mod_act)
             changing_M=0;
       break;
    //-------------------------
       case cmdC3: // change speed [Hz/s]
          Hz_s = RSRx[6]*256 + RSRx[7];
          Mod_s=50/Hz_s;
       break;
       //-------------------------
       default: //error
         TxErr1 = TxErr1+1;
       break;
       }
 /*******************************************/
 }
```

```
 }
}
/*** end of MAIN *****************************************************/


/*********************************************************************
* Function: Inner Loop Timer 4 Interrupt Service Routine every 100us
* Description: InnerLoop called from DefaultIsr.c
*********************************************************************/
void InnerLoop(void)
{
       GpioDataRegs.GPFCLEAR.bit.GPIOF11 = 1; // test Apr 2012
       CalRP = EvaRegs.T2CNT; // Read Encoder Rotor Position
  //-----------------------------------------
       // Slow change Frequency
       if(changing_F!=0){
    Freq_act += (coef_Freq*changing_F*Hz_s*timeInter/1000);
    if((changing_F==1 && Freq_act>FREQ) || (changing_F==-1 && Freq_act<FREQ) ){
                  Freq_act=FREQ;
                  changing_F=0; }
          elements = 1000/(timeInter*(Freq_act/coef_Freq));
          jump_Int = (int)(Tot_elem/elements);
    jump_Rest = (Tot_elem/elements)-jump_Int;
      }
  // Slow change Modulation Index
  if(changing_M!=0){
    Mod_act += (changing_M*(Mod_MAX/Mod_s)*timeInter/1000);
    if((changing_M==1 && Mod_act>MOD_IND) || (changing_M==-1 && Mod_act<MOD_IND) ){
      Mod_act=MOD_IND;
      changing_M=0; }
  }
       //-----------------------------------------
  // {IN} AI1 input voltage [0-4095]
  // {OUT} GPIOF13 switch chopper [open/close]
  DanCount+=1;
  if(DanCount>=30){ // 3[ms]
    DanCount=0;
    if(AI1>Thr_Gen) {
    if(AI1>Thr_Down) {
       CountPowTot += 1; }
    if(Direct_Thr==0) {
       CountPowGen += 1; }
    if(Direct_Thr==1 && AI1>Thr_Up){ //going up & V>(threshold up)
       GpioDataRegs.GPFSET.bit.GPIOF13 = 1; //close switch
       Direct_Thr=0; }
    if(Direct_Thr==0 && AI1<Thr_Down){ //going down & V<(threshold down)
       GpioDataRegs.GPFCLEAR.bit.GPIOF13 = 1; //open switch
       Direct_Thr=1;
```

```
        CountPowTot += 1;
        Power=(VoltagePow*CountPowGen/Resist)*(10*VoltagePow/CountPowTot)/10;
        CountPowTot_s1=CountPowTot;
        CountPowGen_s1=CountPowGen;
        CountPowTot=0;
        CountPowGen=0; }
  }
  else{
     GpioDataRegs.GPFCLEAR.bit.GPIOF13 = 1; //open switch
     Direct_Thr=1;
     CountPowTot=0;
     CountPowGen=0;
     CountPowTot_s1=0;
     CountPowGen_s1=0;
     Power=0;
  }
} //-----------------------------------------

      /*----- Fetch value from Look-Up Table ---- */
      /*--- Phase 1 ---*/
   phase1_Point += jump_Int;
   phase1_Rest += jump_Rest;
   if (phase1_Rest >= 1){
      phase1_Rest -= 1;
      phase1_Point += 1;
   }
   if (phase1_Point > (Tot_elem-1)){
      phase1_Point = phase1_Point - Tot_elem;
        }
   IRt1 = IrefPhase_H1875_L2040[phase1_Point];
   xSin1 = IRt1*Mod_act/Mod_MAX;
   DutyC1 = max + xSin1;

   /*--- Phase 2 ---*/
   phase2_Point += jump_Int;
   phase2_Rest += jump_Rest;
   if (phase2_Rest >= 1){
      phase2_Rest -= 1;
      phase2_Point += 1;
   }
   if (phase2_Point > (Tot_elem-1)){
         phase2_Point = phase2_Point - Tot_elem;
        }
   IRt1 = IrefPhase_H1875_L2040[phase2_Point];
   xSin2 = IRt1*Mod_act/Mod_MAX;
   DutyC2 = max + xSin2;

      /*--- Phase 3 ---*/
```

```
   phase3_Point += jump_Int;
   phase3_Rest += jump_Rest;
   if (phase3_Rest >= 1){
      phase3_Rest -= 1;
      phase3_Point += 1;
   }
   if (phase3_Point > (Tot_elem-1)){
      phase3_Point = phase3_Point - Tot_elem;
         }
   IRt1 = IrefPhase_H1875_L2040[phase3_Point];
   xSin3 = IRt1*Mod_act/Mod_MAX;
   DutyC3 = max + xSin3;

        /*------ PWM Output Test ------------ */
        EvaRegs.CMPR1 = DutyC1;
        EvaRegs.CMPR2 = DutyC2;
        EvaRegs.CMPR3 = DutyC3;

//-----------------------------------------
        // POWER ESTIMATOR
//-----------------------------------------
//if(Mod_act>0){
// V_av1 = (double)(DutyC1-DutyC2)*((AI1-2048)/8.5)/3750;
// I_meas1 = -(Iph1-Iph2)/3; //(*342) int
// V_av2 = (double)(DutyC2-DutyC3)*((AI1-2048)/8.5)/3750;
// I_meas2 = -(Iph2-Iph3)/3; //(*342) int
// V_av3 = (double)(DutyC3-DutyC1)*((AI1-2048)/8.5)/3750;
// I_meas3 = -(Iph3-Iph1)/3; //(*342) int

// PowerAux1 += V_av1*I_meas1*10/342; //(*10)
// PowerAux1 += V_av2*I_meas2*10/342; //(*10)
// PowerAux1 += V_av3*I_meas3*10/342; //(*10)

// CountPowEst+=1;
// if(CountPowEst>=Nsample){
// CountPowEst=0;
// PowerEstTot = (int)((PowerAux1)/(10*Nsample));
// if(PowerEstTot<0){
// PowerEstTot = -PowerEstTot;
// PowerEstGener = 1; }
// else {
// PowerEstGener = 0; }
// Nsample = (int)(elements);
// PowerAux1=0;
// }
// }
// else{
// PowerAux1=0;
```

```
// PowerEstTot=0;
// }

//-----------------------------------------
   // MECHANICAL POWER
 RotSpeed = (AI2-2048)*1.025;
 Torque = (2048-AI3)*0.402;
 RotSpeed_aver = (0.3*RotSpeed)+(0.7*RotSpeed_aver);
 Torque_aver = (0.3*Torque)+(0.7*Torque_aver);
 if(Torque_aver<0){
    Generating = 0; }
 else {
    Generating = 1; }
    PowerMec = 0.105*Torque_aver/10*RotSpeed_aver/100;
//-----------------------------------------

/*------ DAC Output Test Code ---------*/
 if (AOCon2 == 2)
 {
  GpioDataRegs.GPASET.bit.GPIOA12 = 1;  /* DAC CS (July 08) */
  GpioDataRegs.GPACLEAR.bit.GPIOA14 = 1; /* LOADDAC Pulse Low */
  DelayUs(10);
  GpioDataRegs.GPASET.bit.GPIOA14 = 1;
  GpioDataRegs.GPACLEAR.bit.GPIOA12 = 1; /* DAC CS (July 08) */
  if (AOCon1 == 1) {
    CalCount2A = 2048+xSin1;
    CalCount3 = CalCount2A|DAC_CHA;
  }
  else if (AOCon1 == 2) {
    CalCount2B = 2048+xSin2;
    CalCount3 = CalCount2B|DAC_CHB;
  }
  else if (AOCon1 == 3){
    CalCount2C = 2048+xSin3;
    CalCount3 = CalCount2C|DAC_CHC;
  }
  else if (AOCon1 == 4) {
    CalCount3 = CalCount2|DAC_CHD;
  }
  SpiaRegs.SPITXBUF = CalCount3;
  CalCount2 = CalCount2+1;
  if (CalCount2 > 4094)
    CalCount2 = 0;
  AOCon1 = AOCon1+1;
  if (AOCon1 > 4)
    AOCon1 = 1;
  AOCon2 = 0;
 }
```

```
 AOCon2 = AOCon2+1;

/*---------- Read Digital Inputs ----------------------*/
  CalDigIn1 = GpioDataRegs.GPFDAT.all;
  CalDigIn1 = CalDigIn1&0x0700;
  CalDigIn2 = GpioDataRegs.GPEDAT.all;
  CalDigIn2 = CalDigIn2&0x002;
  CalDigIn1 = CalDigIn1>>7;
  CalDigIn2 = CalDigIn2>>1;
  CalDigIn = CalDigIn1|CalDigIn2;
/*******************************************************/

  GpioDataRegs.GPFSET.bit.GPIOF11 = 1;
}


/*************************************************************************
Function: Phase Current Sampling Interrupt Subroutine (every 50us)
This routine stores the ADC samples of all 4 phase current sensors and
all 4 back panel analog inputs into the appropriate location
Description: Called from ADC ISR
*************************************************************************/
void IphSample(void)
{
 static volatile Uint16 GPIOF14_count = 0; // Counter for pin toggle
 GpioDataRegs.GPFCLEAR.bit.GPIOF12 = 1; // test Apr 2012
 PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Must acknowledge the PIE group

/*** Manage the ADC registers ***/
 AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;   // Reset SEQ1 to CONV00 state
 AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;   // Clear ADC SEQ1 interrupt flag

/*** Read the ADC results ***/
 if (ASCB == 1)
 {
   Iph1 = AdcRegs.ADCRESULT0 >> 4; // Read the result
   Iph1 = Iph1 - 0; // remove offset Daniele
   Iph2 = AdcRegs.ADCRESULT1 >> 4; // Read the result
   Iph2 = Iph2 - 0;
   Iph3 = AdcRegs.ADCRESULT2 >> 4; // Read the result
   Iph3 = Iph3 - 0;
   Iph4 = AdcRegs.ADCRESULT3 >> 4; // Read the result
   Iph4 = Iph4 - 40;
   AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 2; // Convert Phase 1 CS
   AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 3; // Convert Phase 2 CS
   AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 4; // Convert Phase 3 CS
   AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 5; // Convert Phase 4 CS
```

```
  ASCB = 2;
 }
 else
 {
  AI1 = AdcRegs.ADCRESULT0 >> 4; // Read the result
  AI1 = AI1 - 20; // remove offset
  AI2 = AdcRegs.ADCRESULT1 >> 4; // Read the result
  AI2 = AI2; // Tweeker
  AI3 = AdcRegs.ADCRESULT2 >> 4; // Read the result
  AI3 = AI3 - 20;
  AI4 = AdcRegs.ADCRESULT3 >> 4; // Read the result
  AI4 = AI4 - 40;
  AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0; // Convert Analog Input 1
  AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 8; // Convert Analog Input 2
  AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 1; // Convert Analog Input 3
  AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 9; // Convert Analog Input 4
  ASCB = 1;
      }

/*** Example: Toggle GPIOF14, which is connected to the LED on the eZdsp board ***/
 if(GPIOF14_count++ > 20000) // Toggle slowly to see the LED blink
             GPIOF14_count = 0;      // Reset the counter
 GpioDataRegs.GPFSET.bit.GPIOF12 = 1; // test Apr 2012
}
/*****************************************************************************/
```

Bibliography

[1] Freris and Infield, *RENEWABLE ENERGY IN POWER SYSTEMS.*

[2] Manwell, McGovan, and Rogers, *WIND ENERGY EXPLAINED, Theory Design and Application.*

[3] H. Blinder, P. A. C. Rosas, R. Teodorescu, and F. Blaabjerg, *Stand alone Gaia-Wind*, September 2004.

[4] H. Li, K. L. Shi, and P. McLaren, "Neural network based sensorless maximum wind energy capture with compensated power coefficent," *Industry Applications Conference. 39th IAS Annual Meeting. Conference Record of the 2004 IEEE*, vol. 4, pp. 2600 – 2608, 2004.

[5] B. K. Bose, *POWER ELECRONICS AND MOTOR DRIVE, Advances and Trends.* USA: Elsevier Inc, 2006.

[6] S. M. Muyeen, *WIND ENERGY CONVERSION SYSTEMS, Technology and Trands.*

[7] Simoes and Farret, *ALTERNATIVE ENERGY SYSTEMS, Design and Analysis with Induction Generator.*

[8] F. Blaabjerg, M. Liserre, and K. Ma, "Power electronics converters for wind turbine systems," *Energy Conversion Congress and Exposition (ECCE)*, pp. 281–290, 2011.

[9] R. Pena, R. Cardenas, R. Blasco, G. Asher, and J. Clare, "A cage induction generator using back to back pwm converters for variable speed grid

connected wind energy system," *Industrial Electronics Society. IECON '01. The 27th Annual Conference of the IEEE*, vol. 2, pp. 1376 – 1381, 2001.

[10] E. Muljadi, K. Pierce, and P. Migliore, *A Conservative Control Strategy for Variable-Speed Stall-Regulated Wind Turbines*, February 2000.

[11] P. K. Banerjee and M. Arifujjaman, "Development of a test-rig for large scale wind turbine emulation," *Electrical and Computer Engineering (ICECE), International Conference on*, pp. 159 – 162, 2010.

[12] T. Hardy and W. Jewell, "Emulation of a 1.5mw wind turbine with a dc motor," *Power and Energy Society General Meeting, IEEE*, pp. 1 – 8, 2011.

[13] W. Li, D. Xu, W. Zhang, and H. MA, "Research on wind turbine emulation based on dc motor," *Industrial Electronics and Applications. ICIEA 2007. 2nd IEEE Conference on*, pp. 2589 – 2593, 2007.

[14] C. Cossar, M. Popescu, T. J. E. Miller, M. McGilp, and M. Olaru, "A general magnetic-energy-based torque estimator: Validation via a permanent-magnet motor drive," *IEEE Transactions on Industry Applications*, vol. 44, no. 4, pp. 1210 – 1217, 2008.

Acknoledgements

Thank you very much to Professor Calum Cossar, always close and full of important advices in the challenge of this Thesis project.
My best thanks to Professor Silverio Bolognani for the know how shared and the accurate preparation in the “Electrical Drivers”.
My thanks also to Peter and Ian, a very helpful team in the SPEED laboratory.
A regard to all my friends, and to my family

and a big kiss to my wife Diane and to my son Samuel.