Comp3402 Wind Turbine Control (coders)

C.B.Price November 2021

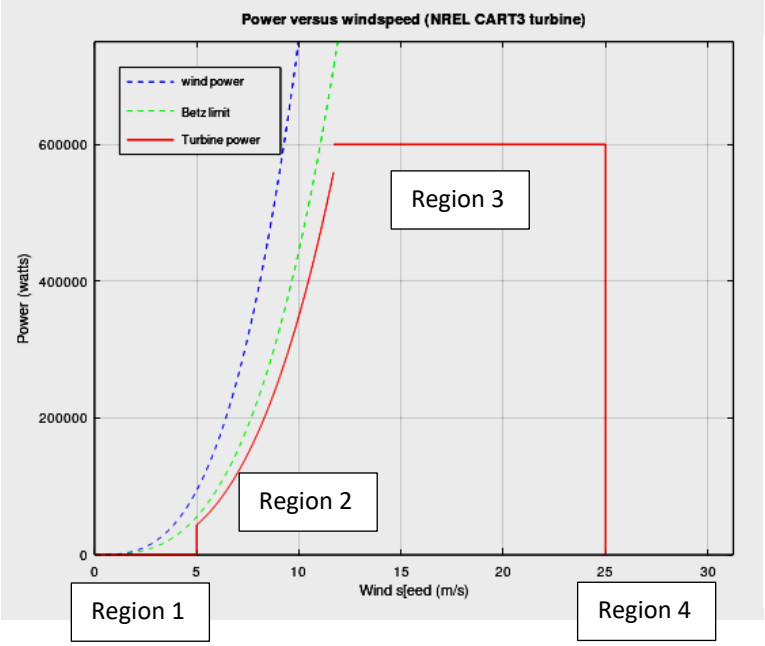
Purpose	(i) To learn about Wind Turbine control strategies for Regions 1-4 with a focus on Region 2 and Region 3. (ii) To conduct a series of **investigations** and subject the observations to analysis. (iii) To have a skirmish with Unreal4 and C++.
Files Required	Unreal4 resources and Octave scripts.
ILO Contribution	LO 4
Send to Me	nix
Homework	Read chapter 4

Investigations

---

1 Parameters for the NREL-CART3 600kW turbine

Below are the parameters for this wind turbine, defined in Unreal and also Octave. On the left is the power curve showing the four regions, the energy in the wind and the Betz limit. These curves are all proportional to wind-speed cubed, i.e. v^3.



R	Rotor radius	20m
ρ	Air density	1.2 kg/m^3
J	Inertia	613,301
λ_{opt}	Optimal tip speed ratio	6.537
C_{pmax}	Maximum power coefficient	0.453
v_{cutIn}	Cut-in wind speed	5 m/s
v_{cutOut}	Cut-out wind speed	25m/s
$v_{cutRated}$	Rated wind speed	11.7 m/s

Perhaps the most fundamental concept we need is λ the tip-speed ratio. This is the ratio of the speed of the tip of the blades as they move through the air, to the wind speed. Of course, the tip speed is in a plane normal to the wind speed:

$$\lambda = \frac{\omega R}{v}$$

Where *R* is the radius of the rotor and ω the angular velocity of the blade in radians per second. You can get this from the rotor rpm using the following relationship

$$1\,rpm \equiv (60)(2\pi) rad/sec$$

---

2 Work through activities 2 and 3 on the non-coders worksheet to gain a basic understanding of turbine control

3 Add an enumerator for various wind source types

The existing code defaults to a fixed wind speed, there is a box **useSteppedWindSpeed** which when ticked gets the code to call a function to step the wind speed. We want to extend this to have three options: steady speed, stepped speed and a time-series of speeds (which model actual measured wind speeds).

(a) Remove the tick box. In **Turbine.h** remove the lines **UPROPERTY(EditAnywhere)** and **bool useSteppedWindSpeed = false;**

(b) Still in the header file, add the following enumeration at the top of the file, after the **#include "Turbine.generated.h"** and before the macro **UCLASS()**

```
UENUM()
enum WindType {
   steady    UMETA(DisplayName = "Steady"),
   stepped   UMETA(DisplayName = "Stepped"),
   series    UMETA(DisplayName = "Time Series"),
};
```

You can choose your own field names (steady, stepped,…) and text which appears in the editor (the UMETA macro text)

(c) Still in the header file declare a variable like **windType** using the enum like this. Put it in the **public:** area of **vars exposed to editor**

```
UPROPERTY(EditAnywhere)
    TEnumAsByte<WindType> windType;
```

(d) Now let's use this new variable. In **Turbine.cpp** look for **ATurbine::Computation()** and change the line referring to the variable **useSteppedSpeeds**, which we have removed, and replace it with the correct enum. Mine would be

```
if(windType == stepped) windV = SteppedWindSpeed();
```

(e) Compile and make sure you have a selection box in the Editor. Quickly test out both options.

4 Code the function to generate a time-series of wind speeds

In **Turbine.c** look for the function **ATurbibe::CalcWindSpeed(double averageSpeed)**. This is completely ineffective and we must replace it. Theory, presented in Chapter 4 tells us how to proceed.

(a) First, given the mean wind speed μ we need to calculate the parameter *c* using the following expression

$$\mu = \frac{c\sqrt{\pi}}{2}$$

So, solve this expression for *c* and code it. You get maths functions as statics like this **FMath::Sqrt(pi)** where **pi** has been declared and assigned in the header file. Remember **aveageSpped** comes in as a function parameter.

The actual speed we shall calculate comes from

$$v = c\sqrt{-\ln n}$$

where *n* is a random number between 0 and 1.

(b) Generate a random number using a call to **FMath::RandRange(0.0f,10f);**

(c) Now code the above expression for *v* and return it from your function.

(d) Now we can add this new option into **ATurbine::Computation()** I chose to do it like this

```
if(windType == series && itn % 100 == 0)
    windV = 5.1 + CalcWindSpeed(averageWindSpeed);
```

Two things. First, I chose to make the call every 100 loops of the computation. Also, I chose to add 5.1 so the speed never drops below the turbine cut-in speed.

(e) Check that **averageWindSpeed** is declared in **Turbine.h**, if not then this is how to do this

```
UPROPERTY(EditAnywhere)
    double averageWindSpeed = 5.0;
```

which sets the default value to 5.0;

(f) You may want to replace to magic number **100** with an editable variable. Now compile and run your code.

5 Clean up the Computation() function

This code in a bit of a mess and you might want to consider re-writing it! In any case we need the following enhancements

(a) Because of the fluctuating wind speed, the controller may cause the turbine to stop rotating, I.e. **omega** (rotational speed) goes to zero. The turbine is now dead so needs restarting.

Add code to test that **omega** is zero (or close) and restart the turbine, the variable **omegaInit** is declared and assigned.

You may wish to print a message to the screen. Here's how to do it

```
GEngine->AddOnScreenDebugMessage(-1, 1.0f, FColor::Red,
   FString::Printf(TEXT(">>>>>>> Turbine Restart")));
```

The **-1** means messages will accumulate and scroll, **+1** will turn this off. The **1.0f** means the message

will remain on the screen for 1.0 seconds

(b) It would be useful to print a message when the turbine changes its region of operation. You could test like this

```
if(region != prevRegion)
```

then print your message; you can use the c-like syntax in your **FString** like this.

```
FString::Printf(TEXT(">>>> Transit from Region %d to region %d
<<<<<<<<<<<"),prevRegion,region))
```

Also, you will need to declare **prevRegion** in the header file.

(c) There is a problem in the code where the turbine changes from Region 3 (where the blades have pitched) since the pitch angle **beta** remains at the Region 3 value. So in **case 2:** we need to gracefully return **beta** to its Region 2 value. We do this incrementally like this

```
beta += 10*(region2Beta - beta) * dT;
```

Again, I seem to have used a magic number here. I wonder what the effect of changing this will be?

(d) In **case 4:** the Turbine is brutally stopped. This can't happen, even with the brake on it will take some time to spin down. We can simulate this dynamics rather like we adjusted **beta** above. The following line will make the turbine spin down to rest

```
omega -= omega * dT / 5.0;
```

The magic number 5.0 has effectively the dimensions of time. So a larger value will mean that the turbine takes longer to spin down.

6 Perform Investigations on wind speed time-series