Jack Felag
PHYS 256
Project Report

Optimization of Wind Turbine Blades

1 Introduction

Wind energy is the topic of some controversial debates because opponents say it is not efficient or cost worthy, as some repairs are just as expensive as building a new turbine. Being a lover of renewable energy, this project seemed to be a good fit for my interests: programming, optimization, and the environment. The fundamental idea was to use an evolutionary algorithm to find better blade shapes, which would cause more rotations, and thus more energy produced as a result. This was all done using vertical-axis wind turbines, as these spin horizontally, so it's easy to visualize because wind and the turbine can both be viewed in the same plane.

2 The Physics

To actually simulate this, I implemented my own physics calculations. Everything was done by hand, except for vector operations and graphics. To actually calculate the torque on the wind turbine, I gave each air particle a very small mass, then multiplied it by its speed to get momentum. Then, using the angle between the air movement vector and the blade segment along with the distance from the point of rotation, got the torque.

$$\tau = \vec{r}\vec{F} = rF\sin(\theta)$$

$$rF\sin(\theta) \approx \sqrt{(x_{air} - x_{center})^2 + (y_{air} - y_{center})^2} * m_{air}v_{air}\sin(\theta)$$

To avoid making the calculation any more complex, I did not give the turbine any mass, so I let the angle of rotation be equal to the approximate torque value. So, there was no mass (therefore, no friction,) on the turbine, so I used the radioactive decay model on its speed to simulate a force of friction. On each time step, the speed of the turbine was multiplied by 66.667%.

The last calculations were done on the reflections of the wind particles. On a simpler coordinate system, we all know the angle of reflection is the same as the angle of incidence. However, that gets complicated when dealing with the the different vectors in the simulation. So, the reflection vector for the particle is:

$$v_{refl} = \vec{v}_{air} - (2(\perp \hat{p}_{segment})(\vec{v}_{air} \cdot (\perp \hat{p}_{segment}))$$

Where v is the velocity vector, and p is the position vector.

3 Implementation

I used a particle-based simulation to implement this. For one man for one semester, A fluid dynamics engine is far out of reach, and wind can be approximated with particles. For the actual code, I created different classes:

- Rotor: Calculates Bezier curve and applies a rotation matrix based on number of blades to create the turbine fan.

- Turbine: Holds an instance of rotor, has a fitness and age, and holds functions for the physics calculations for evaluation.
- Simulator: Has pygame functions to draw the simulation in real-time.
- Population: Holds a number of turbines, contains functions for evolutionary algorithm.

The outline of my simulation (for one turbine) is as follows:

1. Initialize: Puts particles randomly on the screen, draws initial turbine.
2. Iterate:
   (a) Rotate Blades: Applies rotation matrix and has new blade positions.
   (b) Draw Blades: Draw the blades, and puts the blade positions into the form: (x_1, y_1, x_2, y_2)
   (c) Update Wind: Increment each particle's position by its velocity. Checks for collisions, and reflects if there is one.
   (d) Check Boundaries: If a particle is outside of the window, move to $x = 0$ and a random y with the default velocity.
3. Calculate Fitness: Sums all rotation angles at each time step and converts it to number of full rotations.

4 Evolution

To optimize the turbines, I used Age-Fitness Pareto Optimization (AFPO.) This selects for agents with low age, and a high fitness. This produces a Pareto front, which contains agents along the following descriptions: high age high fitness, average age average fitness, low age low fitness. This essentially gives younger agents a quasi-handicap, so it does not necessarily discard them immediately. Say a toddler in your family comes up to you and challenges you to a race. You (hopefully) aren't going to start with them and sprint as fast as you can. Instead, you can either have them start closer to the finish, or you can jog. This levels out the playing field in a way, and gives everyone a decent chance. However, the chance here is time to mutate and get better.

5 Analysis

A single simulation has three main factors: Evaluation time (t), number of particles (n), and number of blade segments (m). This gives us a rough time complexity of $O(tnm)$. However, $m = \{20, 30\}$ because the number of blades are either 2 or 3, and each blade is approximated with 10 segments. That means this is essentially constant, giving a run-time of $O(tn)$. For what it is, this simulation had decent run-time because I had no intra-particle interactions. To include this, the run-time would instead be $O(tn^2)$, because there would be $\binom{n}{2} = O(n^2)$ interactions at each time step.

6 Future Work

To improve on this simulation, I would not go with either fluid dynamics, or intra-particle interactions. It would be nice to see air particles bumping off of each other, but I don't think that would greatly affect the simulation itself. My reasoning for this is air is not very dense, less so at higher altitudes, where turbines are placed.

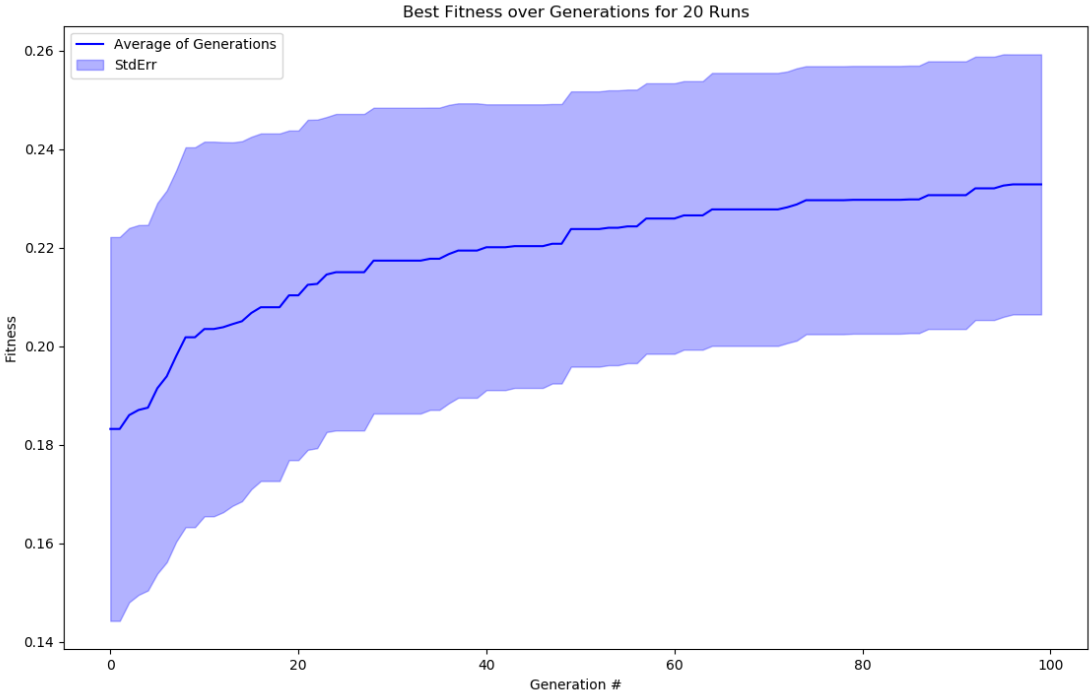
I think a good next step would be involving diffusion of air particles, to make it slightly more

realistic. This would involve discretizing the simulation area into a grid, calculating the number of particles in each grid, then applying a small force on each particle in a high-concentration area towards a low-concentration neighbor.

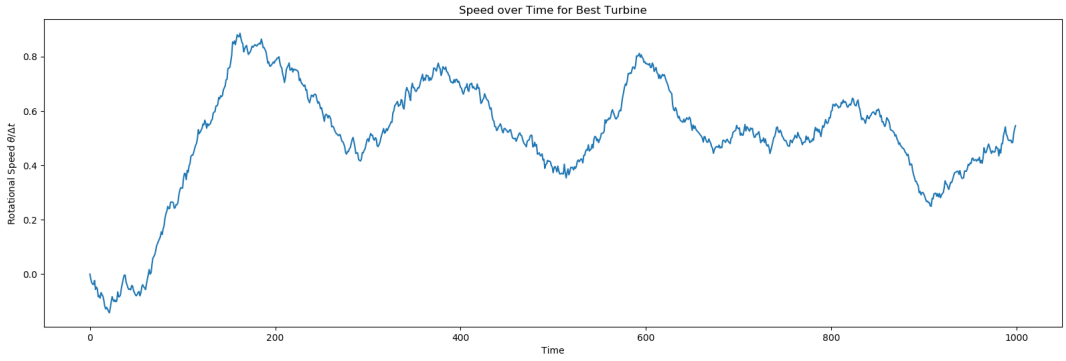
7 Results

Results from before are located in my presentation slides, but for this report, I have the recent results from when I fixed a bug that caused a performance slow down. The initial parameters are also included in the slides, so for this new run, I used 1000 particles, 1000 time steps, and 20 runs of the genetic algorithm.

Average fitness over time for the 20 runs of AFPO.



Speed for the best turbine (degrees per second.)



Drawing of the best turbine with Bezier Curve reference points.