**Top Force and Speed Method**

**Variables:**

step (seconds): Time step between iterations

total\_time (seconds): Used for creating long enough arrays

wheel\_radius (meters): Radius of wheel of bike

gearing (ratio): gear ratio from motor to wheel

rider\_mass(kg): Mass of rider

bike\_mass(kg): Mass of bike

gravity(m/s^2): Acceleration of Gravity

air\_resistance or drag coefficient(dimensionless): resistance of air considering shape of bike

air\_density(kg/m^2): density of air

frontal\_area(m^2): Frontal cross sectional area of the bike, the area of the bike the air will hit

rolling\_resistance(dimensionless): coefficient of friction given to material of tire and ground

top\_torque(Nm): top amount of force the motor can output.

top\_rpm(rpm): top rpm of motor

efficiency: percentage of energy used that does not go to actual movement of bike

**Logic Flow:**

1) Time:

Step time by time step

2) Distance:

Determine current distance given time step and previous speed

If over max distance end simulation

3) Look Up Speed:

Look up speed using a distance to speed look up table

For IOM use distance to speed table of a faster bike

4)Top Speed Check:

Check if look up speed is higher than top speed of simulated bike.

If higher than stop speed make speed equal to top speed

5) Force:

Calculate force necessary for bike to apply to ground to overcome forces (drag, rolling resistance, inclination, acceleration)

6)Top Force Check:

Check if force is higher than top force of simulated bike

If higher find speed that gives top force and set force equal to top force

7) Power:

Calculate power from force, speed, and efficiency

8) Energy

Calculate Energy (KW/hour) from power and time step

**Code:**

notation: [n] means previous step [n+1] means current step

1)Time

*time[n+1] = time[n] + step*

2)Distance

*distance[n+1] = distance[n] + speed[n]\*step*

3) Look up

*l\_speed[n+1] = distancetospeed\_lookup(distance[n+1])*

4) Check To Speed

*if l\_speed[n+1] > top\_speed:*

*t\_speed[n+1] = top\_speed*

*else:*

*t\_speed[n+1] = l\_speed[n+1]*

5) Force

*acceleration[n+1] = (speed[n+1] - speed[n])/step*

*drag[n+1] = 0.5 \* drag\_area\*air\_density\*speed[n+1] \*\*2*

*altitude[n+1] = distancetoaltitude\_lookup(distance[n+1])*

*slope[n+1] = (altitude[n+1] - altitude[n])/(distance[n+1] - distance[n])*

*incline[n+1] = mass\*gravity\*slope[n+1]*

*rolling[n+1] = mass\*gravity\*rolling\_resistance*

*force[n+1] = acceleration[n+1] + drag[n+1] + incline[n+1]*

6) Check Top Force

*if c\_force[n+1] > top\_force:*

*#find speed that gives top force*

*speed[n+1] = (opt.fsolve(force\_solve,t\_speed[n+1],n))[0]*

*force[n+1] = Force(speed[n+1],n)*

*else:*

*speed[n+1] = t\_speed[n+1]*

*force[n+1] = c\_force[n+1]*

7) Power

*power[n+1] = (force[n+1] \* speed[n+1])/efficiency*

8) Energy

*energy[n+1] = energy[n] + power[n+1]\*(step/(60\*60))*

Benefits of This Method:

* Using data to look up distance to speed allows for easy way to model turns and break points
* Top speed and Force gives an easy place for the bike's power-train to be added to the model

Problems with Model:

* Relies on data for distance to speed look up a lot. This data must be faster than the teams bike to be correct. This data also doesn't correctly model how Rob would go around IOM or how the bike handles.
* Euler method is not perfect for ODE solving

Future Work:

* Use a better Solver, better than Euler method.
* Include tire's ability to apply force
* Batteries ability to give power required
* Temperature