Physical Simulation Optimization using Q-Learning

Kelton Busby
`busbykt@mail.uc.edu`

Abstract

In this paper a reinforcement learning algorithm called Q-Learning is applied to maximize velocity and minimize driver fatigue in a human powered vehicle attempting a land speed record. Q-Learning results in a set of Q-values for state-action pairs, called a policy. That policy is a function of power input over time for a given driver, vehicle, and environment combination that maximizes velocity for the driver's fatigue limit. This function of power input over time results in higher achievable speeds than other methods of developing a power over time function. Several extensions of the Q-Learning algorithm are explored, along with a multi-driver vehicle simulation result.

1 Problem Background

A yearly competition to break a human powered land speed record takes place in Battle Mountain, Nevada. Competitors attempt to accelerate their vehicle over a 5 mile stretch of road towards a 200 meter section where their speed is calculated. The record for that speed as of May 2021, is 89.59 miles per hour [IHPVA 5]. To reach high speeds the driver must manage fatigue while accelerating. Minimizing fatigue ensures that as much energy as possible is available to the driver to reach a maximum possible speed. So how should one go about managing fatigue? The driver's only control is how much power to input to the pedals. Input power is a combination of force on the pedals and cadence or revolutions per minute (RPM). The result of this input is accumulation of fatigue, and acceleration of the vehicle. As velocity increases, the driver must input more power to accelerate, overcome air drag, and overcome rolling resistance, which increases fatigue. The optimal policy of power input over time is sought using Q-Learning.

The initial Q-Learning algorithm was developed in [Busby 4]. This algorithm was limited by:

- An incomplete physics model.
- A sub-optimal reward function.
- A single driver limit.

Improving the algorithm and simulation in these three ways is the focus of this paper.

2 Governing Physics - Improving the physics Model

Most vehicles attempting human powered land speed records are lightweight aerodynamic shells that roll on bicycle wheels, pedaled by a person in a recumbent position. A vehicle rolling on the ground requires power to overcome rolling resistance, aerodynamic drag, climb a slope if it exists, and accelerate. The power applied to overcome those resistances and accelerate is reduced by mechanical losses within the drivetrain of the vehicle. The power equation is:

$$P_{total} = \frac{P_R + P_D + P_C + P_A}{\eta}$$

Where P_R is power to overcome rolling resistance, P_D is power to overcome aerodynamic drag, P_C is power to climb, P_A is power to accelerate, and η is drivetrain efficiency. Each power on the right hand side can be broken down further into its constituent parts:

$$P_R = vmgC_{rr} \qquad P_D = \frac{1}{2}\rho v^3 AC_D \qquad P_C = vmg * sin(arctan(s)) \qquad P_A = vma$$

Resulting in:

$$P_{total} = \frac{v(mg(C_{rr} + sin(arctan(s)) + \frac{1}{2}\rho v^2 AC_D + ma)}{\eta}$$

Where v is velocity, m is mass, g is acceleration due to gravity, C_{rr} is coefficient of rolling resistance. ρ is air density A is vehicle frontal area, C_D is vehicle drag coefficient, s is slope, and a is acceleration. All units are SI units. The power it takes to maintain any velocity v (set $a = 0$) can be solved for, given that the vehicle, environment, and driver properties C_D, C_{rr}, s, m, g,and ρ are known. In this application the power equation is used to solve for acceleration a at discretized time steps where current velocity and road slope are known. The power equation described above improves the previous physical simulation equation by adding the climbing component P_C. Previous simulations were assumed to be on perfectly level courses. Given knowledge about a course, one can further optimize their maximum velocity by accounting for road gradient/slope s. While land speed record tests are on a flat course, more complex courses can be simulated knowing grade.

In addition to calculating the accumulation of velocity over time, a fatigue model is used to approximate the available energy a driver has remaining to propel the vehicle. Fatigue F accumulates as a function of power p and duration of that power input t, according to the following equation:

$$F = \frac{t}{T_{max}(p)}$$

A T_{max} curve represents a relationship between power, and the maximum duration a driver can maintain that power. An example set of curves can be found in Appendix A - *Fatigue Curve* and on page 50 of [Wilson 1]. The fatigue model assumes constant accumulation of fatigue. If a driver can maintain a 400 watt input for 100 seconds and reach full fatigue $F = 100\%$, 10 seconds of 400 watt input accumulates $F = 10\%$ or 10% fatigue.

Finally, there is the calculation of Coefficient of rolling resistance C_{rr}. In this case, it is assumed to be solely a function of velocity v and can be found via the function reproduced in Appendix A - *Rolling Resistance*.

3 Q-Learning - Improvements

The goal in this application of Q-learning is to generate a driver, environment, and vehicle specific policy starting at 0 velocity v and 0 fatigue F. The learned policy should maximize velocity while not exceeding a $F = 100\%$ fatigue threshold. for specifics about the Q-learning algorithm see Appendix B - *Q-Learning Algorithm*. The primary features of Q-learning in this application are the state space, action space, and environment. See Appendix B - *State, Action Space, and Environment* for definitions. The Q-Learning algorithm developed in [Busby 4] is modified in two significant ways. First the reward function is modified to incentivize acceleration instead of velocity, and the random action rate ϵ is no longer constant, but is a function of iteration number in training.

3.1 Training the Agent

With a selection of environment variables, simulations are run to find an optimal policy given the environment. Each simulation starts at state space ($0v$, $0F$) where v is velocity and F is fatigue. The agent chooses an action based on two factors the hyper-parameter ϵ which determines how often the agent takes a random action, and the q-value of the available state-action pairs. If the agent does not act randomly, it chooses the action with the maximum q-value. From here the simulation follows these steps:

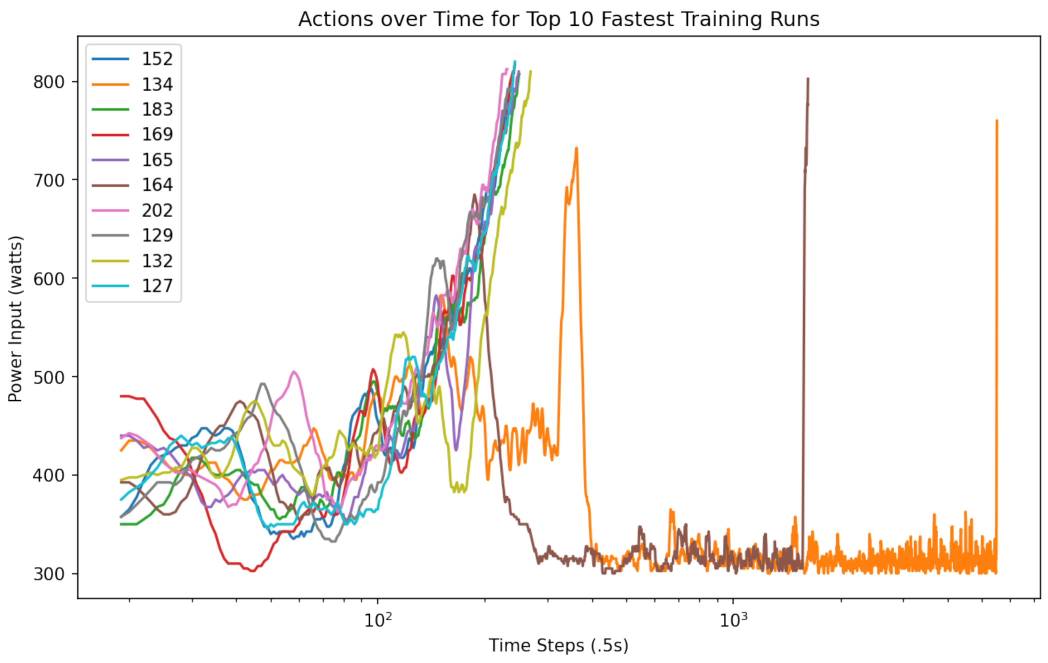

Figure 1: Old reward function training set power over time

- If the current state fatigue F is equal to or greater than 1, corresponding to a fully fatigued driver, the simulation stops.
- Given an action, the next velocity state and fatigue are calculated, see appendix B - *Next Velocity, Next Fatigue*.
- The Environment steps t_s seconds in time.
- Reward is calculated and the q-value of the previous state is updated.
- The current state (v_n,F_n) becomes the next state (v_{n+1},F_{n+1})

In [Busby 4], the reward function was $Reward = \frac{velocity}{fatigue+1}$. With broad discretizations of time, fatigue, and velocity, this occasionally resulted in round-off error that extended the training unrealistically, and sub-optimally. It is possible to fix this with a very fine discretization of timestep, fatigue, and velocity, but at the expense of enormous computational cost for the Q-learning algorithm. Instead, the problem has been resolved with an updated reward function that reduces the training time while improving the resulting policy. The new reward function $Reward = \frac{acceleration}{fatigue+0.0001}$ incentivizes continued increases in velocity, where the old had the potential to get "stuck" choosing low power inputs for increases in fatigue that rounded-off to 0 for small timesteps. See figure 1 orange and brown iterations that extended an order of magnitude or more in time beyond the other iterations. Note the improvement in top 10 fastest training runs between figure 1 and 2.

The algorithm was further updated to include a linearly decaying random-action rate ϵ. Where

$$\epsilon = .8 * max(\frac{n-i}{n}, 0) + .1$$

This formula corresponds to a linearly decreasing likelihood of a random action as training progresses. Starting from 90% each iteration drops the likelihood of a random action being taken until the final simulation has a 10% chance of taking a random action at each state.

3.2 Adding a Second Driver

A second driver requires a more complex simulation environment. The simulation Q-states must track both riders power inputs and fatigue levels. A simplistic approximation of this more complex

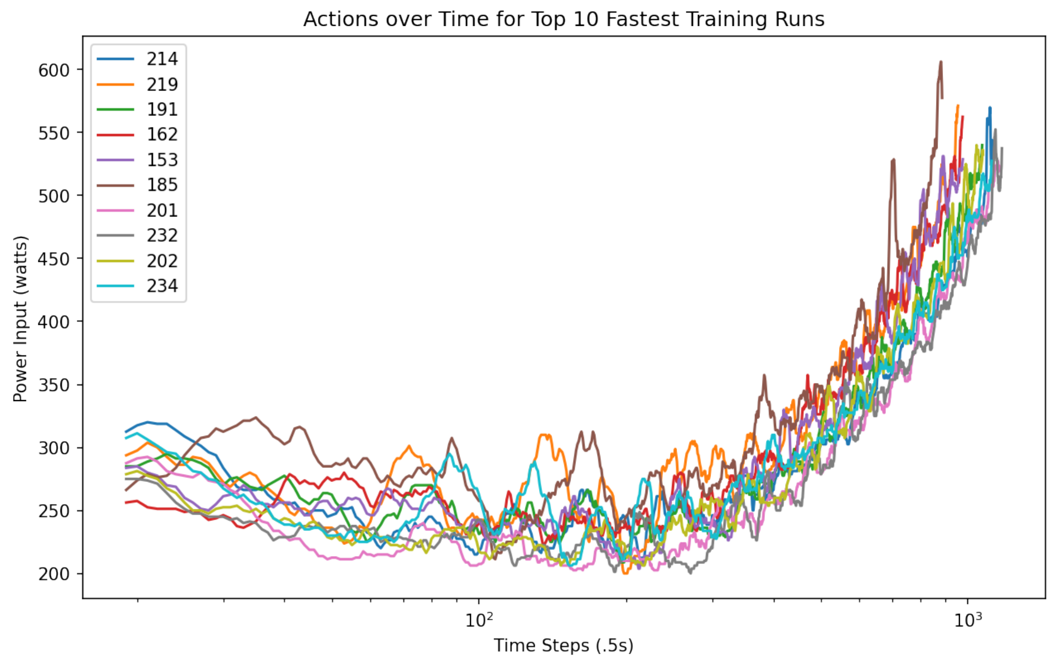

Figure 2: New reward function training set power over time

environment is to double the power input from a single rider. This allows the simulation to retain the single driver computational complexity, but does make the assumption that both riders input equal power at all times. This also assumes that both riders have similar cycling capability as a function of their weight, See Appendix A - *Fatigue Curve*. A second driver also requires new vehicle characteristics (frontal area and drag coefficient) to fit an additional rider. The characteristics can be determined via CFD simulation. See Appendix D - *Tandem Vehicle Definition and Setup*.

4 Results

Three limitations to the Q-Learning algorithm and simulation outlined in [Busby 4] were proposed to address at the beginning of this paper:

- An incomplete physics model.
- A sub-optimal reward function.
- A single driver limit.

The improved physics model allows for road gradient to be considered when optimizing a power-input plan for a given driver and vehicle combination. While this does not effect the flat course for the IHPVA event in Nevada, it does allow for accurate simulation of a more varied course.
The reward function adjustment to seek the greatest acceleration to fatigue ratio instead of velocity to fatigue ratio allows for accurate and fast simulations, even with coarse discretizations of time, velocity, and fatigue. This is an improvement on the older velocity-rewarded model. See section 3.1 - *Training the agent*.
The addition of a second driver was achieved with a slight modification to the single driver simulation and changed setup parameters to accommodate the vehicle and driver characteristics. See the results of the top 10 training runs on the next page.

Figure 3 shows the top ten training runs (as measured by maximum velocity achieved) after 1000 iterations of Q-Learning. Each color represents the power input of each driver over time as they accelerate towards a top speed. The numbers associated with each color is the iteration n. As expected, the top runs are towards at the end of the training effort (n almost 1000) after the reinforcement agent has learned about the environment.

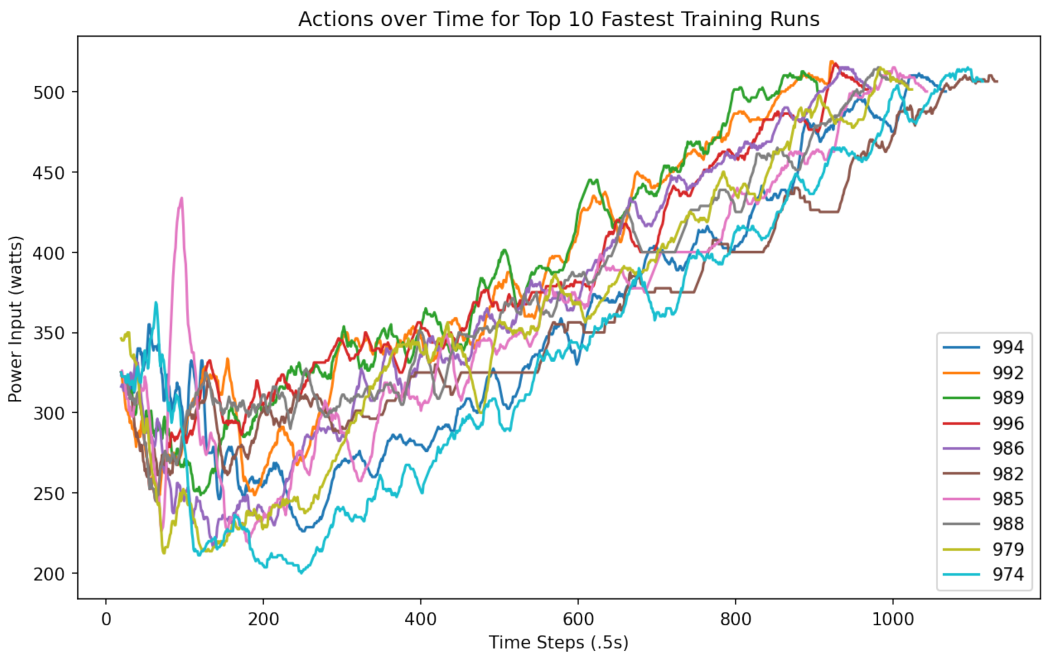

Figure 3: Rolling 10 second mean agent actions over time

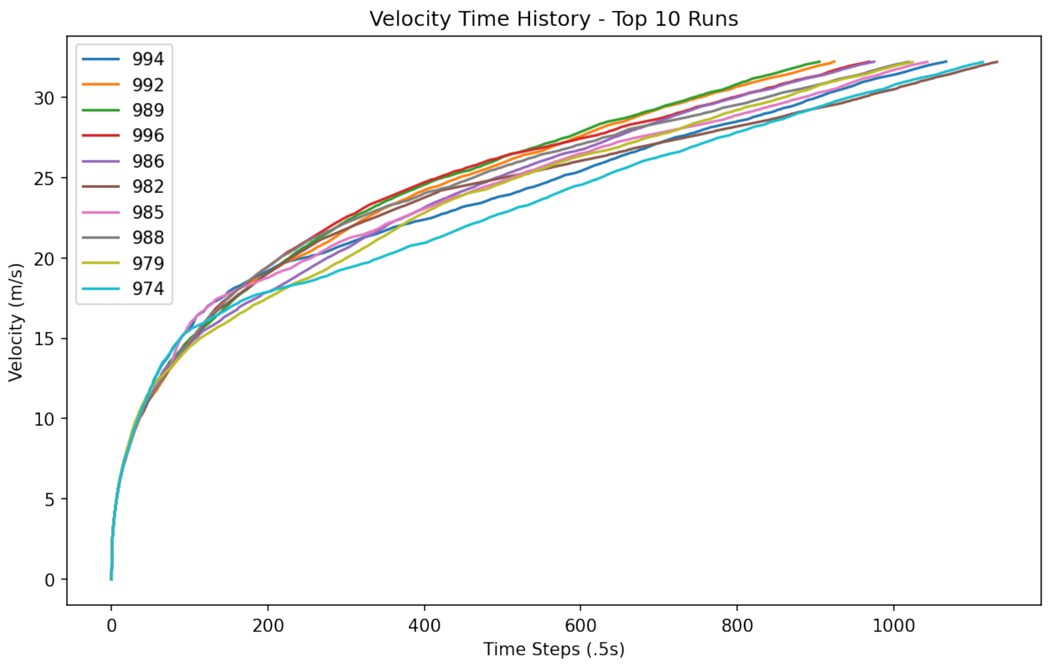

Figure 4: Rolling 10 second mean vehicle velocity over time

Figure 4 shows the same training runs as figure 3, but on axes of velocity over time. Notice how the velocity ramps up quickly, as the power to overcome aerodynamic drag and rolling resistance is low, but as the simulations continue, the acceleration slows dramatically. This is expected behavior, as the algorithm searches for the most fatigue-efficient power for the driver to apply that still accelerates the vehicle.

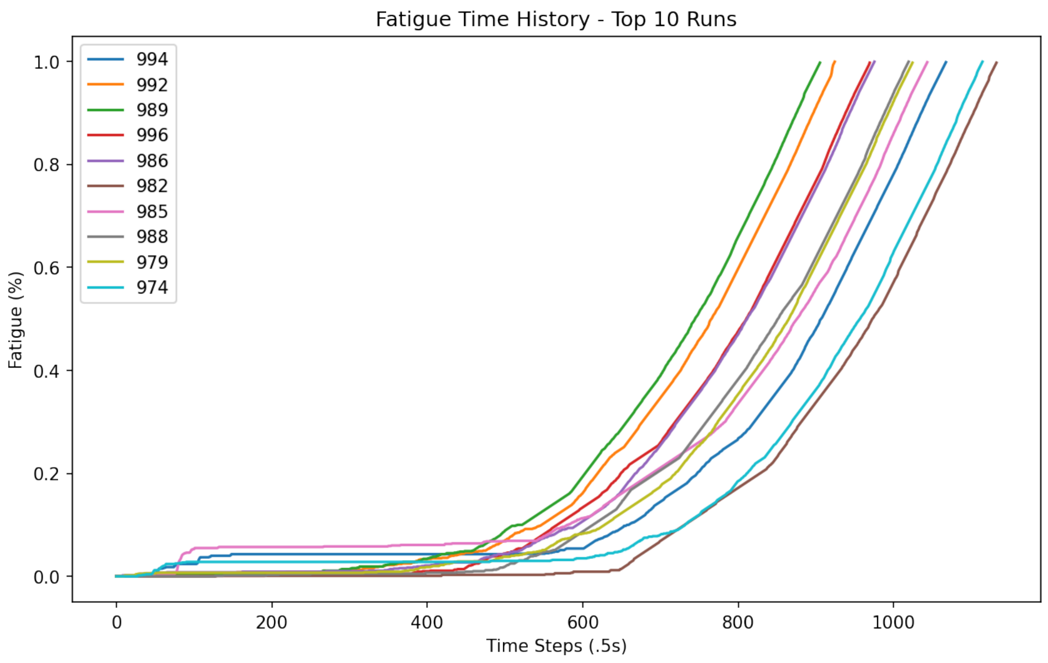

Figure 5: Rolling 10 second mean driver's fatigue over time

In figure 5, again the same training iterations are shown, but on axes of fatigue over time.The fastest runs all wait until fatigue is absolutely necessary to accumulate to continue accelerating the vehicle. The crescendo to a top speed is an incredibly careful pursuit at the highest speeds. Marginal increases in rolling resistance or aerodynamic drag can make the pursuit of a human powered land speed record implausible.

5 Discussion

From the perspective of improving the Q-Learning algorithm and simulation, this exploration has been a success. The ability to simulate optimal power-over-time profiles for individualized rider and environment conditions is a useful tool both in vehicle design and use. The tandem rider exploration has not yet reached its potential. First, the current algorithm only allows for a tandem simulation of two equally capable riders that take equal action at every step. Adding the ability to simulate individually acting riders increase the size of the Q-Learning state space enough to make the training time impractically long on a modern laptop. Regarding the potential to break the current human powered land speed record, an extremely capable rider (or two) and a carefully designed vehicle are the key. The tool developed in [Busby 4] and extended here allow iterating preliminary design and human-capability testing well before a massive investment in building and testing in the real world. The results of a final simulation assuming two 95th percentile cyclists in a tandem NACA 662015 vehicle is shown in appendix E - *95th Percentile Tandem Results*.

References

[1] Wilson, D. G., &amp; Schmidt, T. (2020). Bicycling science (4th ed.). Cambridge, MA: MIT press.

[2] Johnstone, D. (2018, June 7). How does your cycling power output compare? Retrieved 2021, from https://www.cyclinganalytics.com/blog/2018/06/how-does-your-cycling-power-output-compare

[3] Morrison, A. (2010, March 20). AFM tire testing rev9. Retrieved 2021, from http://www.biketechreview.com/tires_old/images/AFM_tire_testing_rev9.pdf

[4] Busby, K. (2021, March 23) Q-Learning for Speed. Retrieved 2021, from kb.shoelace.biz/wp-content/uploads/2021/03/Q-Learning-for-Speed.pdf

[5] "IHPVA - International Human Powered Vehicle Association." Retrieved 2021, from Ihpva.org, ihpva.org/hpvarecl.htmnom01

[6] Ira Herbert Abbott, and Von Doenhoff. Theory of Wing Sections. 2nd ed., Dover Pubns., 60, 1959.

6 Appendix A - Governing Physics

6.1 Fatigue Curve

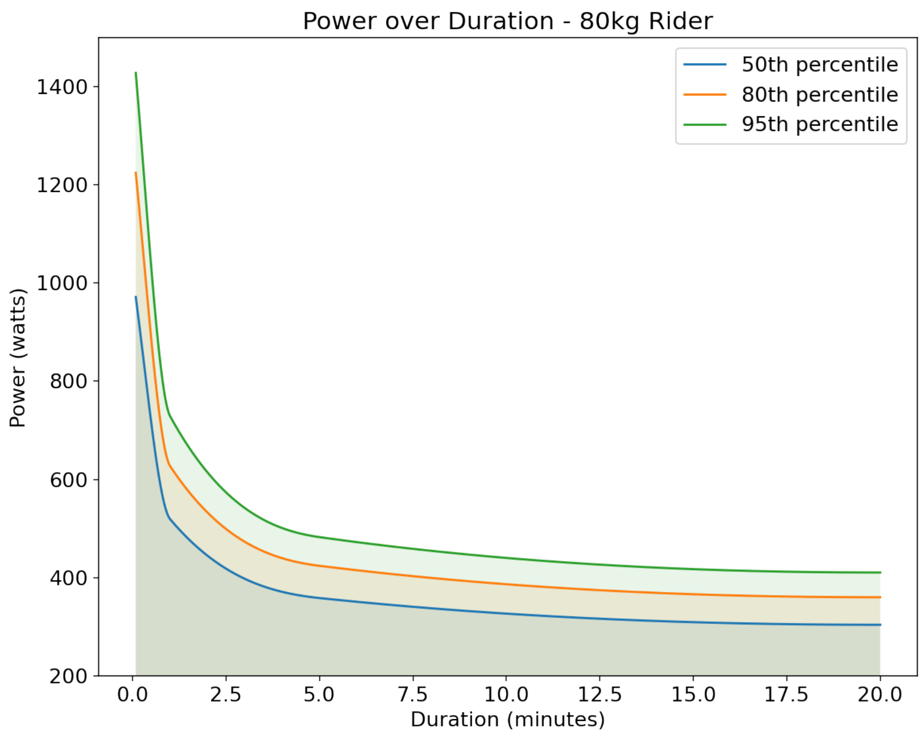

Figure 6: 80kg driver fatigue curves

6.2 Rolling Resistance

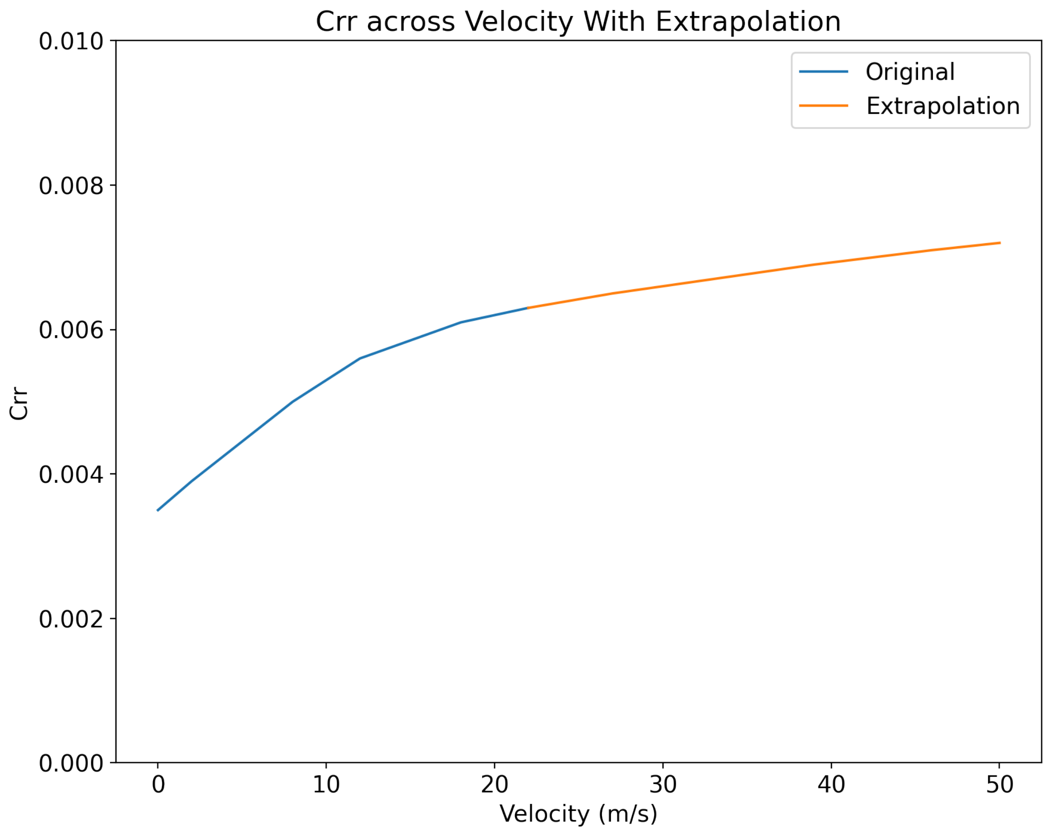

Figure 7: C_{rr} as a function of velocity

7 Appendix B - Q-Learning

7.1 Q-Learning Algorithm

The Q-Learning algorithm is implemented exactly as prescribed in literature.

- Given a current state, take an action according to randomness or maximum Q-value.
- Calculate reward and discount times maximum Q-value of next state-action pairs.
- Update Q-value of current state action pair.

The Q-Learning update incorporated is as follows:

$$Q_{n+1}(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[Reward(s,a,s') + \gamma \max_{a`} Q_n(s,a)]$$

where s is state, a is action, s' is next state and a' is next action.

7.2 State, Action, Environment

A **state** s is comprised of a velocity v and fatigue F pair. This ensures that all relevant information needed to calculate a reward ($Reward = \frac{acceleration}{fatigue+0.0001}$) is encoded. Further, this simple state definition keeps the problem size manageable.

The **action** a space is any power p to be input by the driver for the duration of the next time step. The state and action space must be discretized in such a way to balance accuracy of the simulation, while maintaining a tractable problem size.

The **environment** is the set of factors that govern the simulation results, and are constant throughout the simulation runs. The driver has cycling capability, and weight. The vehicle has weight, frontal area, and drag coefficient. And the external environment has gravity and air density. These are all components of the simulation environment.

7.3 Next Velocity, Next Fatigue

Next velocity is the velocity of the vehicle at the next time step after taking an action at the current time step. It is calculated using the power equation and is implemented in code here:

```
def next_velocity(time_delta, current_velocity, power_in, vehicle,
                  slope, density=1.225):
    # power to roll
    PR = Pr(v=current_velocity, m=vehicle.mass, Cr=Crr(current_velocity),
            eta=vehicle.eta)
    # power to overcome drag
    PD = Pd(p=density,v=current_velocity,A=vehicle.A, Cd=vehicle.CD,
            eta=vehicle.eta)
    # power to climb
    PC = Pc(v=current_velocity, m=vehicle.mass, s=slope, eta=vehicle.eta)
    # power to accelerate
    PA = (power_in - PR - PD - PC)*vehicle.eta
    A = PA/(vehicle.mass*current_velocity)
    NV = current_velocity+A*time_delta
    return NV, PA, PD, PR, PC

def Pd(p,v,A,Cd,eta=.96):
    '''Calculate power to overcome aerodynamic drag.'''
    return .5*p*v**3*A*Cd/eta

def Pr(v,m,Cr,eta=.96):
    '''Calculate power to overcome rolling resistance'''
    return v*m*9.81*Cr/eta

def Pc(v,m,s,eta=.96):
    '''Calculate power to climb a grade(slope).'''
    return v*m*9.81*np.sin(np.arctan(s))/eta

def Pa(v,m,a,eta=.96):
    '''Calculate power to accelerate.'''
    return v*m*a/eta

crr = [.0035,.0039,.005,.0056,.0061,.0063,.0065,.0067,.0069,.0071,.0072]
vs = [0,2,8,12,18,22,27,33,39,46,50]

crr_df = pd.DataFrame(crr, index=vs, columns=['Crr'])
crr_df = crr_df.reindex(np.arange(0,50.01,.01)).interpolate('pchip')
Crr = interp1d(crr_df.index, crr_df['Crr'])
```

Next fatigue is calculated by summing current fatigue with the accumulation of fatigue over the next time step given a power input. Next fatigue is implemented in code here:

```
def fatigue(action,t0,t1,fatigue_curve):

    ts=t1-t0

    return ts/fatigue_curve(action)
```

Note that the fatigue curve is rider dependent. Some are shown in Appendix A - *Fatigue Curve*

8 Appendix C - Results

8.1 Original Setup

Alpha (learning rate)	α	.5
Gamma (discount)	γ	.1
Epsilon (random action rate)	ϵ	.25
Number of Simulations	n	250
Time Step	t_s	$.5s$
driver Mass	m_r	$80kg$
driver Percentile	R_p	80%
Vehicle Mass	m	$20kg$
Gravitational Acceleration	g	$9.81\frac{m}{s^2}$
Air Density	ρ	$1.07\frac{kg}{m^3}$
Vehicle Frontal Area	A	$0.325m^2$
Drag Coefficient	C_D	0.0496
Drivetrain Efficiency	η	0.97
Starting Acceleration	a	$0\frac{m}{s^2}$
Starting Velocity	v	$0\frac{m}{s}$
Starting Fatigue	F	$0\frac{m}{s}$
Coefficient of Rolling Resistance	C_{rr}	See Fig 4.

8.2 Improved Setup Adjustments

Epsilon (random action rate)	ϵ	$.8 * max(\frac{n-i}{n}, 0) + .1$

9 Appendix D - Tandem Vehicle Definition and Setup

The tandem vehicle characteristics should minimize drag, while allowing for enough room within the vehicle to house two equally sized cyclists (178cm tall, 80kg). The vehicle outer profile simulated here is a NACA 662015 symmetric airfoil [Abbott 6] rotated around its longitudinal axis to make a volume. This liberally approximates a tandem vehicle, albeit without the additional drag from wheel wells or other protrusions like a forward facing camera. The 662015 foil was chosen due to its favorable lift and drag characteristics at 0 degrees angle of attack, along with its thickness at 15% of the chord length being suitable for drivers to fit into the vehicle. The vehicle frontal area is calculated

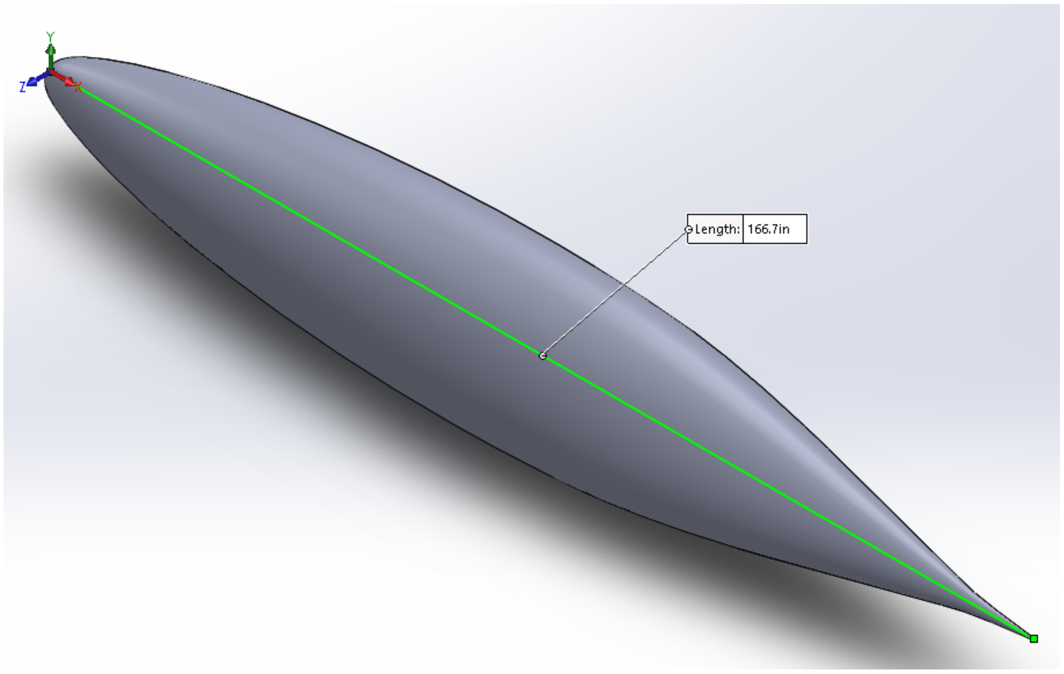

Figure 8: The NACA 662015 symmetric volume

by knowing that the thickness is 15% of the chord length of 166.7 inches, or 25 inches in diameter. The frontal area is circular, with a maximum radius of 12.5 inches. Therefore the frontal area A is $0.317m^2$. Assuming that the wheel wells and other necessary protrusions will increase the frontal area, a more conservative estimate to simulate is 0.35 m^2.

The drag coefficient is determined by performing an external flow CFD simulation on the volume. The simulation ran with free stream velocity of 60 mph along the longitudinal axis (x) of the volume and a fluid density of $1.225\frac{kg}{m^3}$, resulting in a force on the vehicle of $9.22N$. The vehicle drag coefficient is calculated from the following formula:

$$F_d = \frac{1}{2}\rho v^2 A C_d$$

Where F_d is drag force, ρ is air density, v is velocity, A is frontal area, and C_d is drag coefficient. Solving for drag coefficient:

$$9 = \frac{1}{2}1.225(26.8224^2)(.31)C_d$$

$$C_d = .064$$

The following is the tandem simulation setup:

Alpha (learning rate)	α	$.5$
Gamma (discount)	γ	$.1$
Epsilon (random action rate)	ϵ	$.8 * max(\frac{n-i}{n}, 0) + .1$
Number of Simulations	n	1000
Time Step	t_s	$.5s$
Driver Total Mass	m_r	$160kg$
Driver Capability Percentile	R_p	80%
Vehicle Mass	m	$35kg$
Gravitational Acceleration	g	$9.81\frac{m}{s^2}$
Air Density	ρ	$1.07\frac{kg}{m^3}$
Vehicle Frontal Area	A	$0.35m^2$
Drag Coefficient	C_D	0.064
Drivetrain Efficiency	η	0.94
Starting Acceleration	a	$0\frac{m}{s^2}$
Starting Velocity	v	$0\frac{m}{s}$
Starting Fatigue	F	$0\frac{m}{s}$
Coefficient of Rolling Resistance	C_{rr}	See Fig 4.

Notice adjustments in driver and vehicle mass, drag coefficient, and drivetrain efficiency when compared to the single driver setup.

10 Appendix E - 95th Percentile Tandem Results

Simulation setup for two 95th percentile riders in a tandem NACA 662015 symmetric airfoil volume vehicle:

Alpha (learning rate)	α	.5
Gamma (discount)	γ	.1
Epsilon (random action rate)	ϵ	$.8 * max(\frac{n-i}{n}, 0) + .1$
Number of Simulations	n	1000
Time Step	t_s	$.5s$
Driver Total Mass	m_r	$180kg$
Driver Capability Percentile	R_p	95%
Vehicle Mass	m	$35kg$
Gravitational Acceleration	g	$9.81\frac{m}{s^2}$
Air Density	ρ	$1.07\frac{kg}{m^3}$
Vehicle Frontal Area	A	$0.35m^2$
Drag Coefficient	C_D	0.064
Drivetrain Efficiency	η	0.94
Starting Acceleration	a	$0\frac{m}{s^2}$
Starting Velocity	v	$0\frac{m}{s}$
Starting Fatigue	F	$0\frac{m}{s}$
Coefficient of Rolling Resistance	C_{rr}	See Fig 4.

This resulted in a final Q-learned agent top speed of 79.22 mph.