Vehicle Physics and Tire Dynamics in Just Cause 4

Hamish Young
Lead Mechanics Designer, Avalanche Studios

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19


JUST™
CAUSE 4

Goals: Open-World-Action Physics

Limited CPU Budget

30Hz Timestep

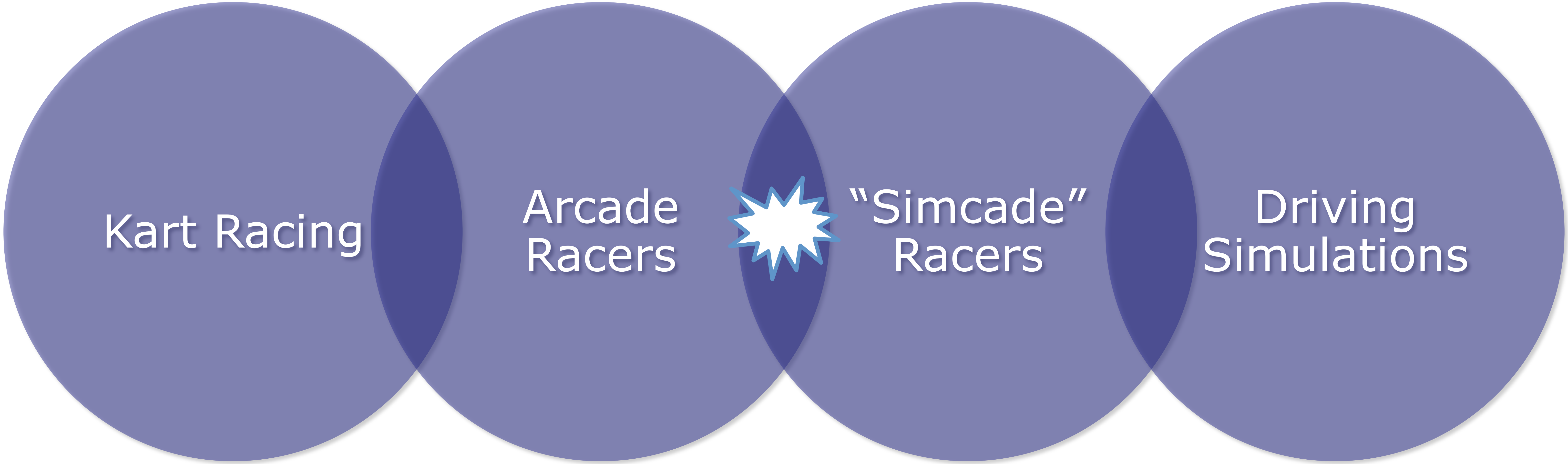
“Believable”

Limited “Cognitive Load”

Wide Range of Vehicles

Diverse Driving Environment

Range of handling



Just Cause 4 recipe

- Similar input parameters as simulation models
- Higher grip than real (especially in braking phase)
- Friction clamps to stay physically stable
- Drawn friction curves
- Scale down pitch and roll components
- Add “driver assists” e.g. drift control on whole vehicle

MF-Tire and semi-empirical models

1. Take a real tire.
2. Measure forces in a machine with varying input parameters.
3. Parameterize so mathematical formulae curve-fit forces.

Requires real tire data: hard to hand-modify

Real tires have undesirable properties

Poor feedback at 30Hz especially with game pads

- Wheel load sensitivity causes transient behavior.
- Using weight transfer for cornering becomes unreliable.

Understeer under braking

- Requires too much planning for open world action game.

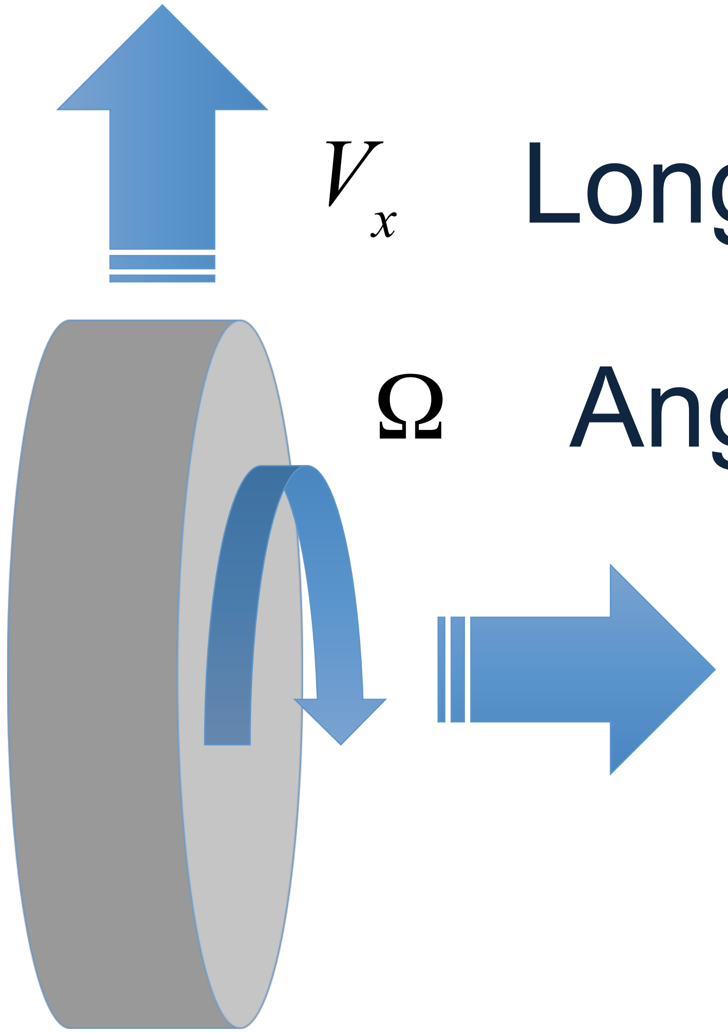
Oversteer can be corrected by traction control and stability control

- Indirect control is complicated to get right.

Tire setup

- Wheel position and orientation (incl. steer)
- Wheel linear velocity
- Wheel angular velocity
- Tire ground patch position and normal

Tire reference frame



V_x Longitudinal wheel velocity

Ω Angular velocity of wheel

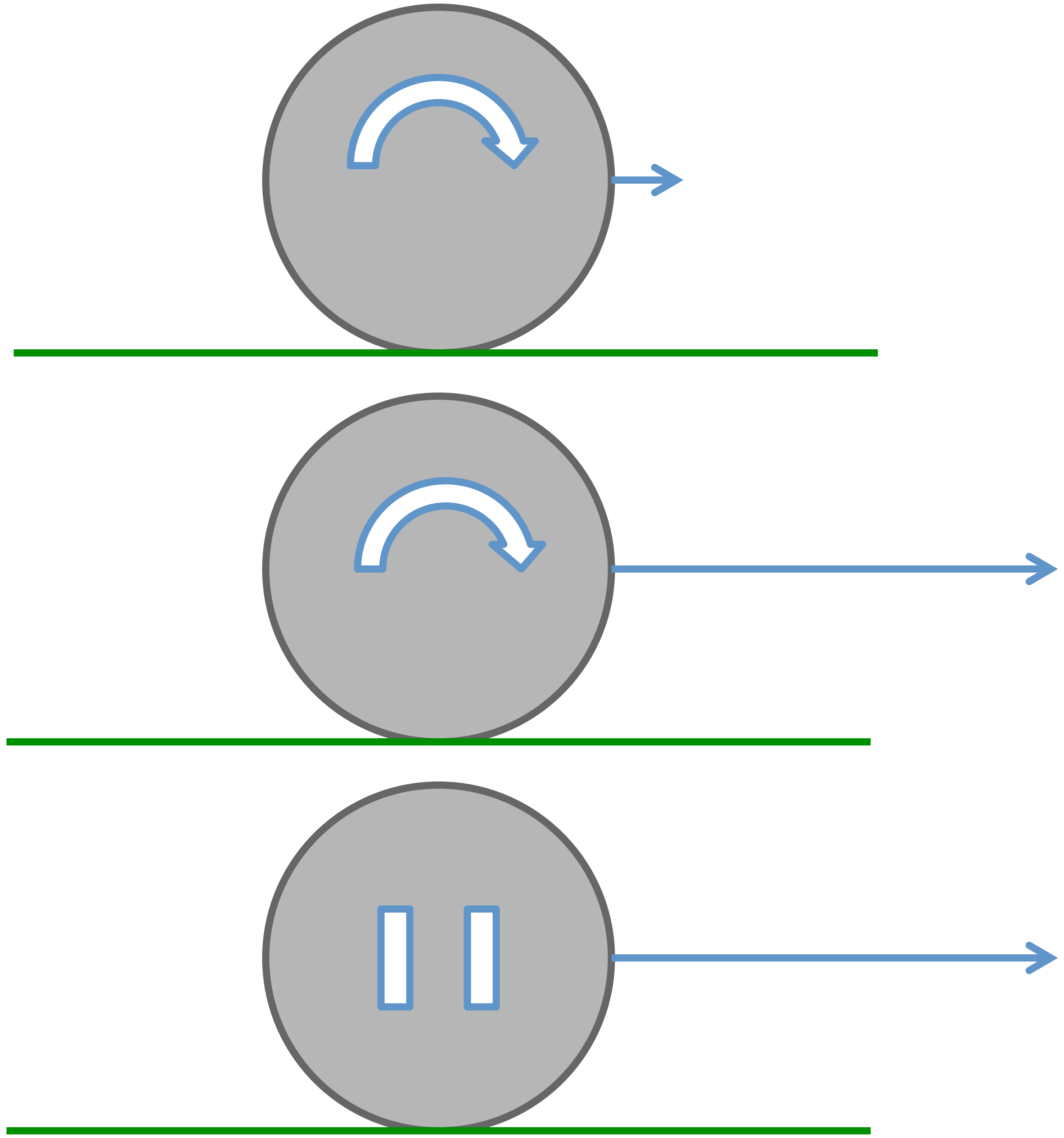
V_y Lateral wheel velocity

Slip Ratio

Wheel spin:

Rolling:

Locked:



Slip Ratio = +ve

Slip Ratio = 0

Slip Ratio = -1

Input parameters: Slip Ratio

$$SlipRatio = \frac{\Omega r}{V_x} - 1$$

Ω	Angular velocity of wheel
r	Wheel radius
V_x	Longitudinal wheel velocity

```
float longitudinal_wheel_speed_ms = wheel_contact_velocity_relative_to_ground.dot(wheel_forward_dir);

float wheel_slip_ratio_SAE = ((wheel_angular_velocity * wheel_radius) / longitudinal_wheel_speed_ms) - 1.0f;
```

Gotcha: Slip Ratio

$$SlipRatio = f(\Omega, V_x)\frac{(\Omega r - V_x)}{|V_x|}$$

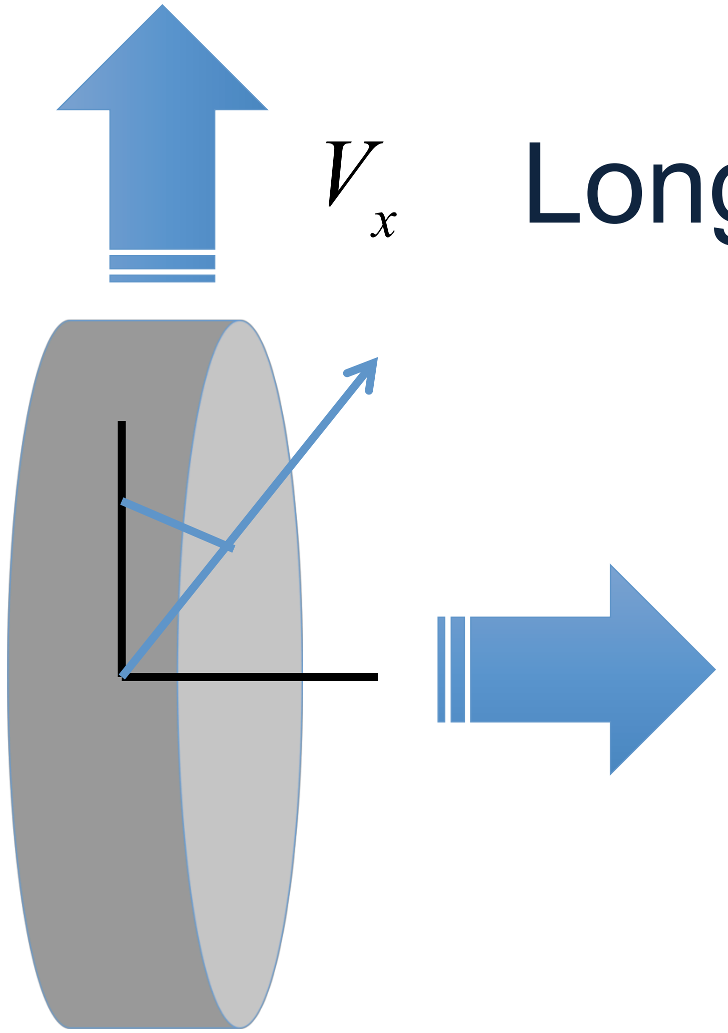
```
float longitudinal_wheel_speed_ms = wheel_contact_velocity_relative_to_ground.dot(wheel_forward_dir);

// Work whether wheel angular velocity is reliable for its sign direction
bool is_wheel_stopped = abs(wheel_angular_velocity) < kEpsilon;

// When wheel is locked / stopped – slide direction (+/-1.0f) comes from the wheel speed
float slide_sgn = is_wheel_stopped ? Signf(longitudinal_wheel_speed_ms) : Signf(wheel_angular_velocity);

float wheel_slip_speed_ms = ((wheel_angular_velocity * wheel_radius) - longitudinal_wheel_speed_ms) * slide_sgn;
float wheel_slip_ratio_SAE = wheel_slip_speed_ms / abs(longitudinal_wheel_speed_ms);
```

Slip Angle



V_x Longitudinal wheel velocity

V_y Lateral wheel velocity

Input parameters: Slip Angle

$$SlipAngle = \arctan\left(\frac{V_y}{|V_x|}\right)$$

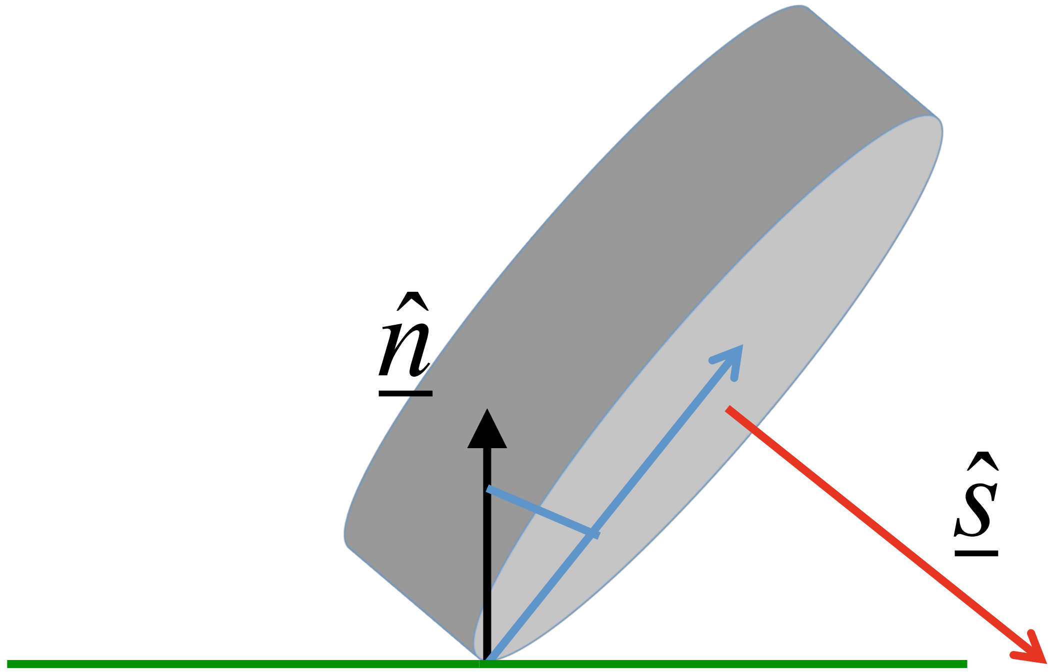
V_x Longitudinal wheel velocity

V_y Lateral wheel velocity

```
float longitudinal_wheel_speed_ms = wheel_contact_velocity_relative_to_ground.dot(wheel_forward_dir);
float lateral_wheel_speed_ms = wheel_contact_velocity_relative_to_ground.dot(wheel_right_dir);

float wheel_slip_angle_rad = atan2(lateral_wheel_speed_ms , abs(longitudinal_wheel_speed_ms));
```

Camber Angle



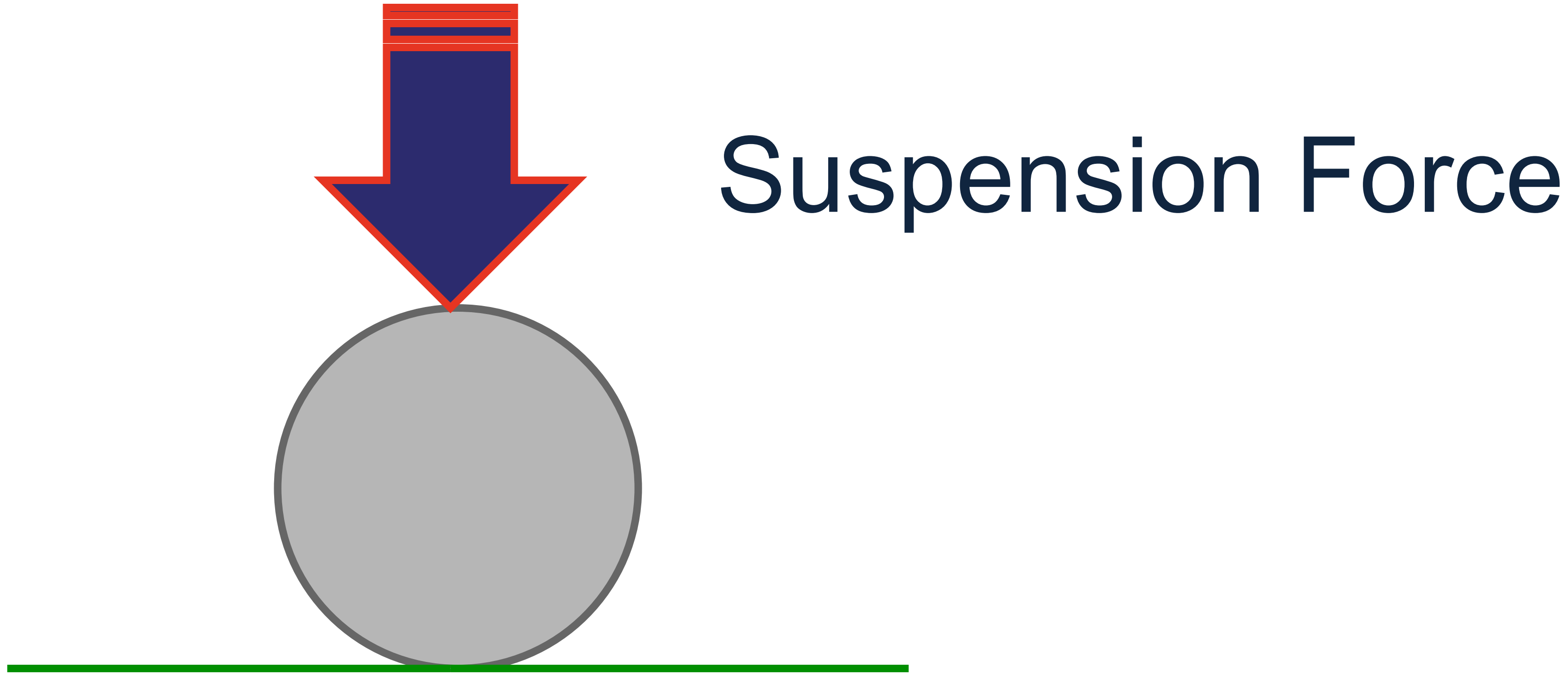
$$CamberAngle = \frac{\pi}{2} - \arccos\left(\hat{\underline{n}} \bullet \hat{\underline{s}}\right)$$

$\hat{\underline{n}}$ Ground Contact Normal

$\hat{\underline{s}}$ Wheel Spin Axis

```
float camber_cosangle = clampf(wheel_contact_normal.dot(spin_axis_world), -1.0f, 1.0f);
float wheel_camber_rad = (PI / 2.0f) - acos(camber_cosangle);
```

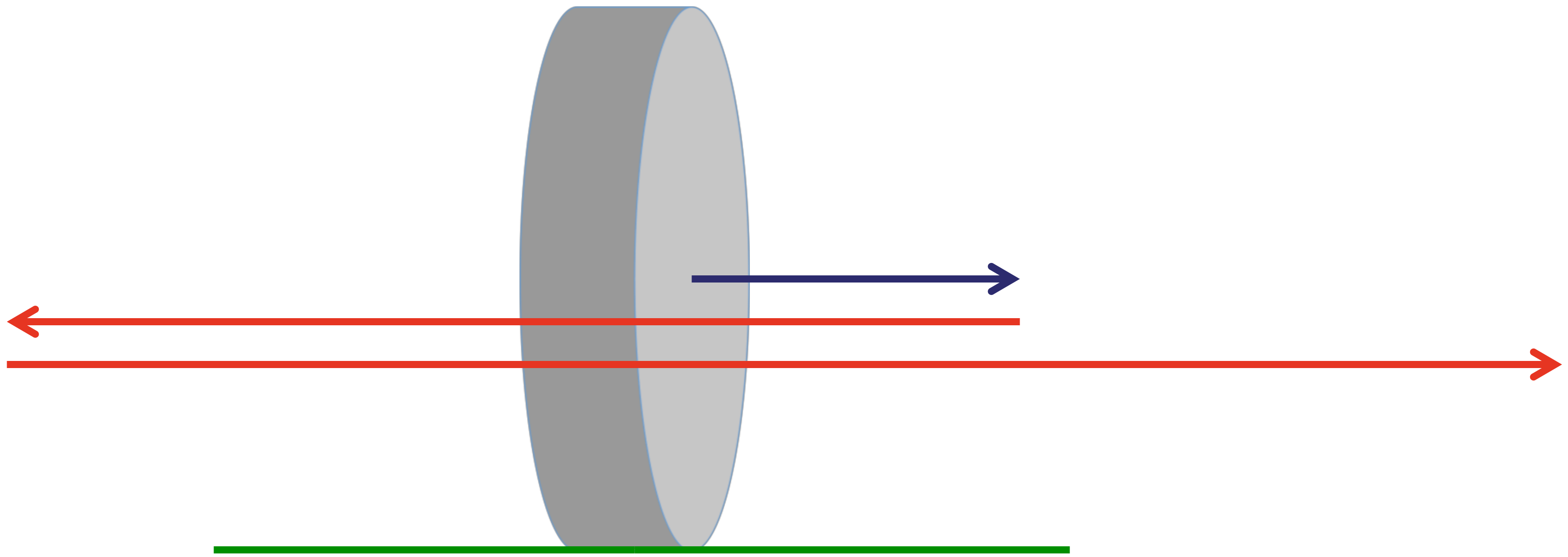
Input parameters: Wheel Load




```
float wheel_load = powf(suspension_force_at_rest,   1.0f - wheel_load_responsiveness) *
                   powf(suspension_force,           wheel_load_responsiveness);
```

Higher than real grip

- Model can diverge:



Meaning the grip (which is a kind of drag) is flipping the sign of the velocity

Friction clamps

- Don't let too much friction force flip the sign of the velocity.

$$Force_{Max} = -\frac{m|\underline{v}|}{timestep}$$

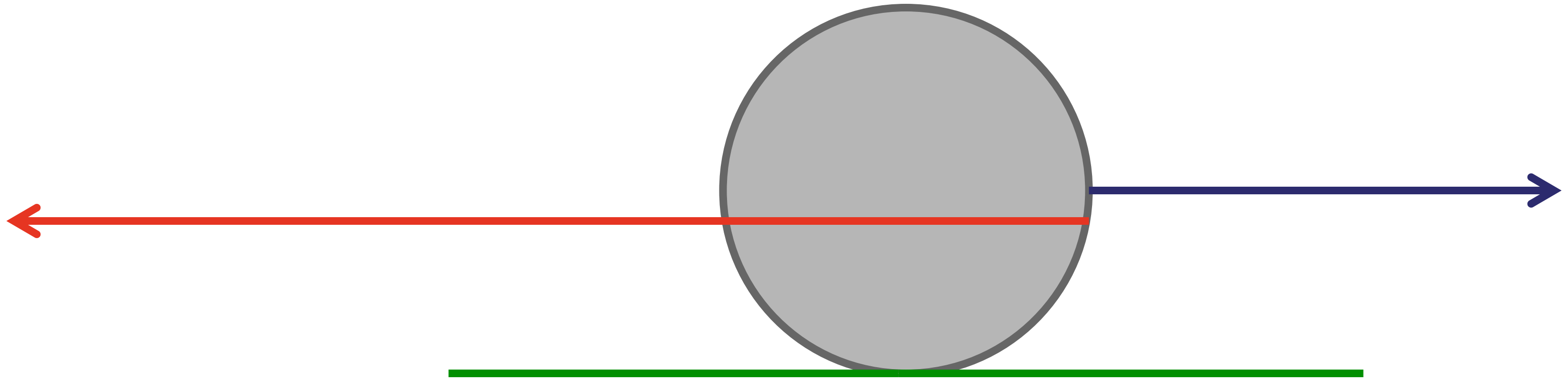
- This is the force required to stop the object in a single timestep.
- Tires are a bit more complicated.

Useful Mass

Scale your clamp per wheel by how much that wheel should contribute

```
float wheel_load_factor = wheel_load / total_wheel_load;
float useful_mass = wheel_load_factor * vehicle_mass;
```

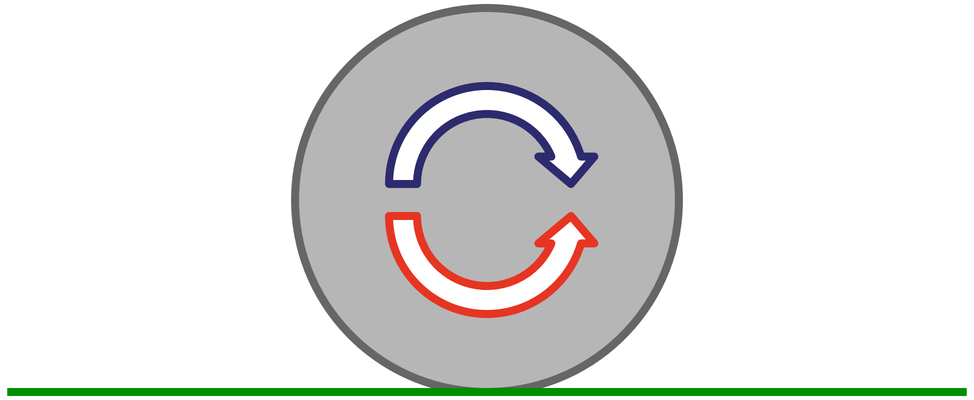
Longitudinal Friction Clamp



```
float max_fwd_force = (useful_mass * wheel_slip_speed_ms / delta_time)
                    + (wheel_torque * slide_sgn / wheel_radius);
```

Angular Clamp

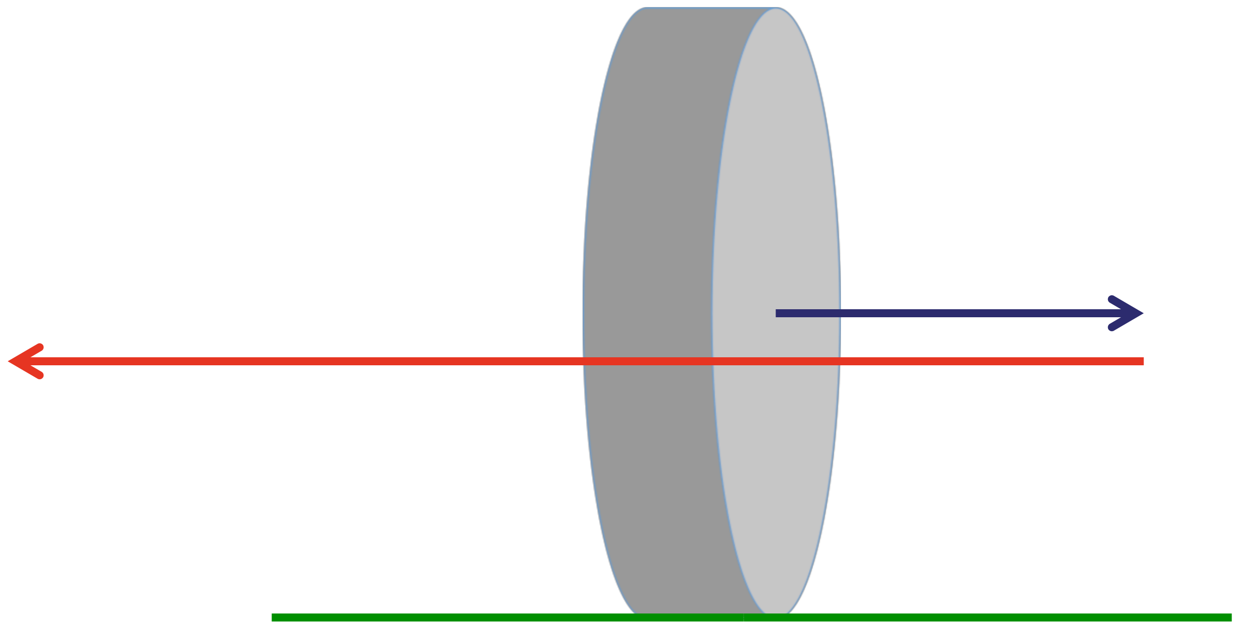
Prevent wheel spinning wrong way relative to the road



```
float estimated_longitudinal_wheel_speed_ms = longitudinal_wheel_speed_ms +
                                              (fwd_force * slide_sgn * delta_time / vehicle_mass);
float estimated_new_road_spin_velocity = estimated_longitudinal_wheel_speed_ms / wheel_radius;

float spin_vel_diff = wheel_angular_velocity - estimated_new_road_spin_velocity;
float spin_friction = (spin_vel_diff / (wheel_inv_inertia * delta_time));
float spin_max_ground_fwd_force = spin_friction * slide_sgn / wheel_radius;
```

Lateral Clamp



Lateral Clamp

Simple?

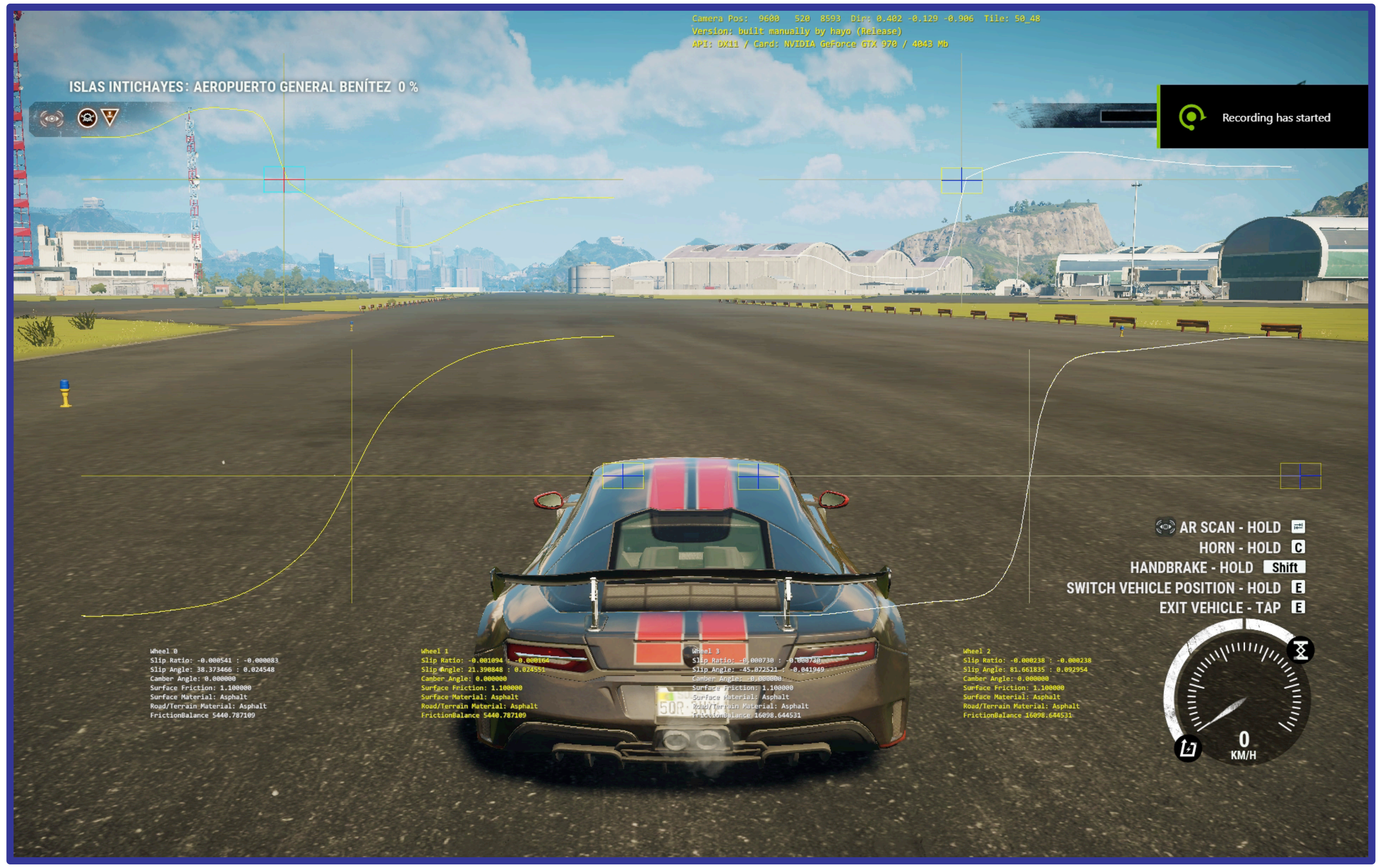
```
float max_right_force = (useful_mass * wheel_speed_right_ms / delta_time);
```

Lateral Clamp

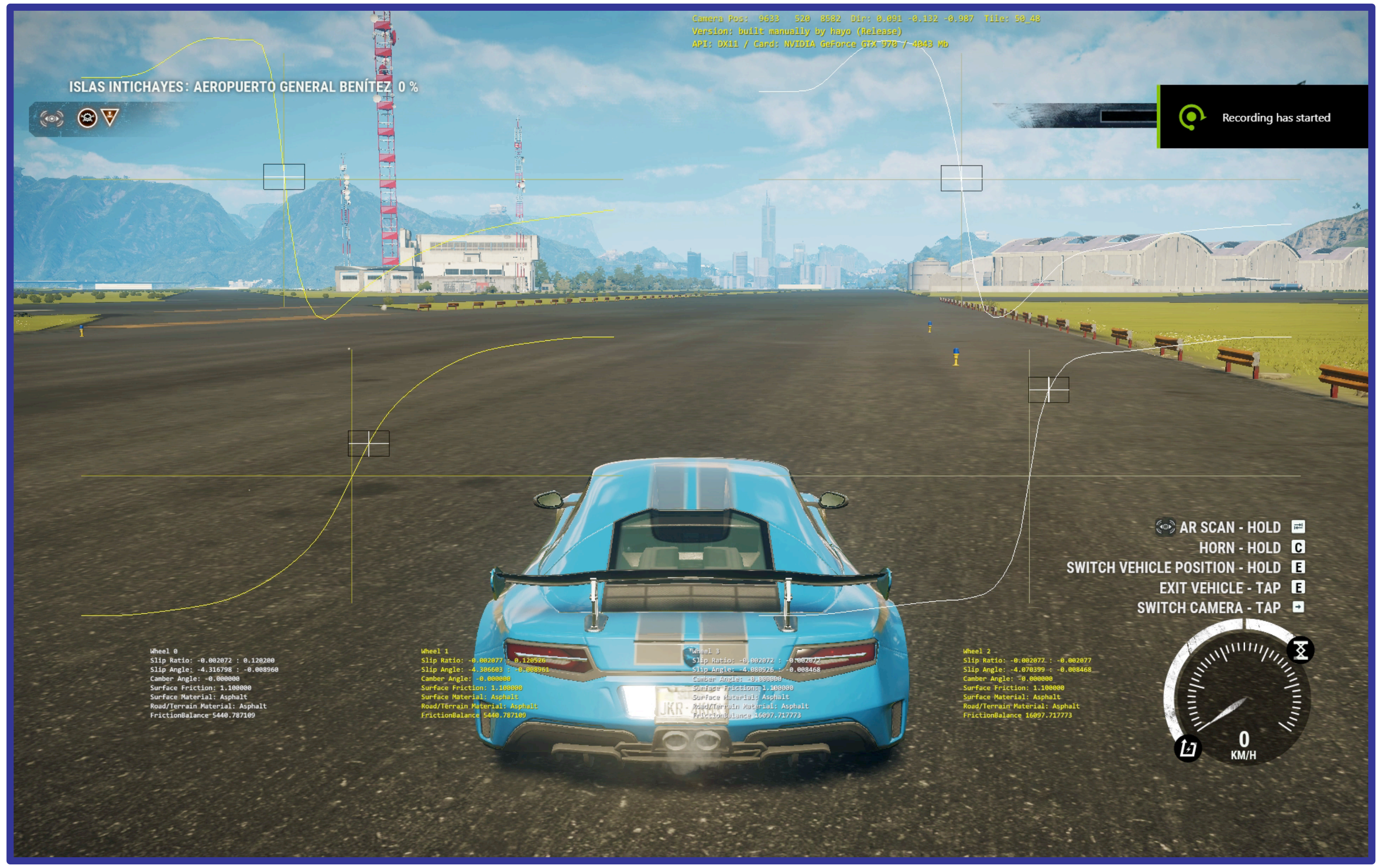
```
float wheel_load_factor = wheel_load / total_wheel_load;
float max_right_force = right_force;
vector wheel_arm = wheel_force_position - vehicle_center_of_mass_in_world;

// Determine the axis of rotation due to the force
vector arm_cross_force = vector::cross(wheel_arm, wheel_right);
vector force_rotation_axis = arm_cross_force.normalize();
if (!force_rotation_axis.isZero())
{
    vector inertia_force_rotation_axis_vector = vehicle_inertia_matrix * force_rotation_axis;
    float inertia_around_force_rotation_axis = abs(force_rotation_axis.dot(inertia_force_rotation_axis_vector));
    vector arm_cross_force_cross_arm = vector::cross(arm_cross_force, wheel_arm);
    float inverse_angular_factor = arm_cross_force_cross_arm.dot(wheel_right) / inertia_around_force_rotation_axis;
    float inverse_mass = 1.0f / vehicle_mass;
    float inertia_at_point = 1.0f / (inverse_mass + inverse_angular_factor);
    // Compare this with mass to see how much the slamp is affected by the rotational component
    max_right_force = - wheel_load_factor * inertia_at_point * wheel_speed_right_ms / delta_time;
}
else
{
    max_right_force = - wheel_load_factor * vehicle_mass * wheel_speed_right_ms / delta_time;
}
```

Friction clamps in action

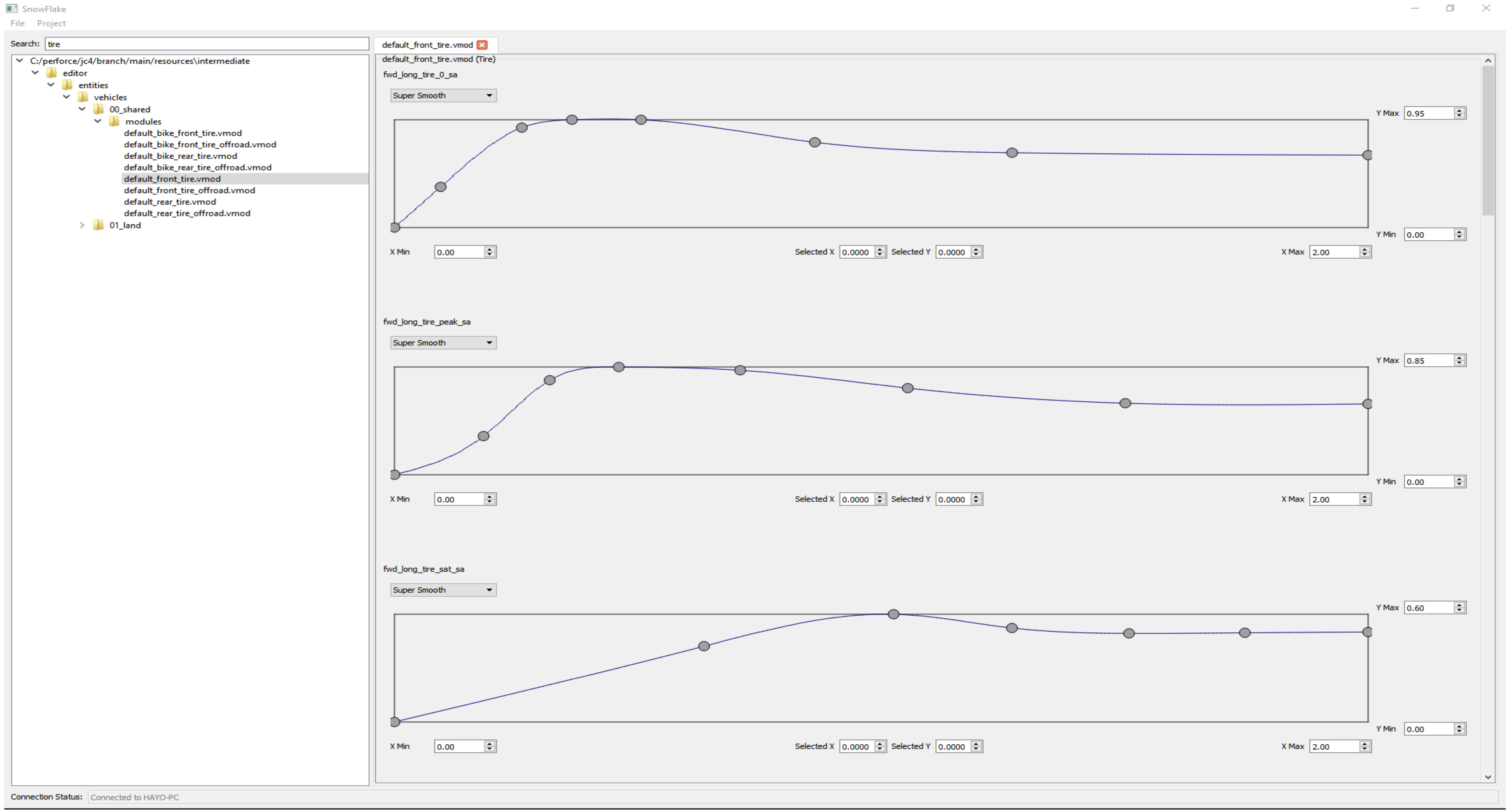


Friction clamp on

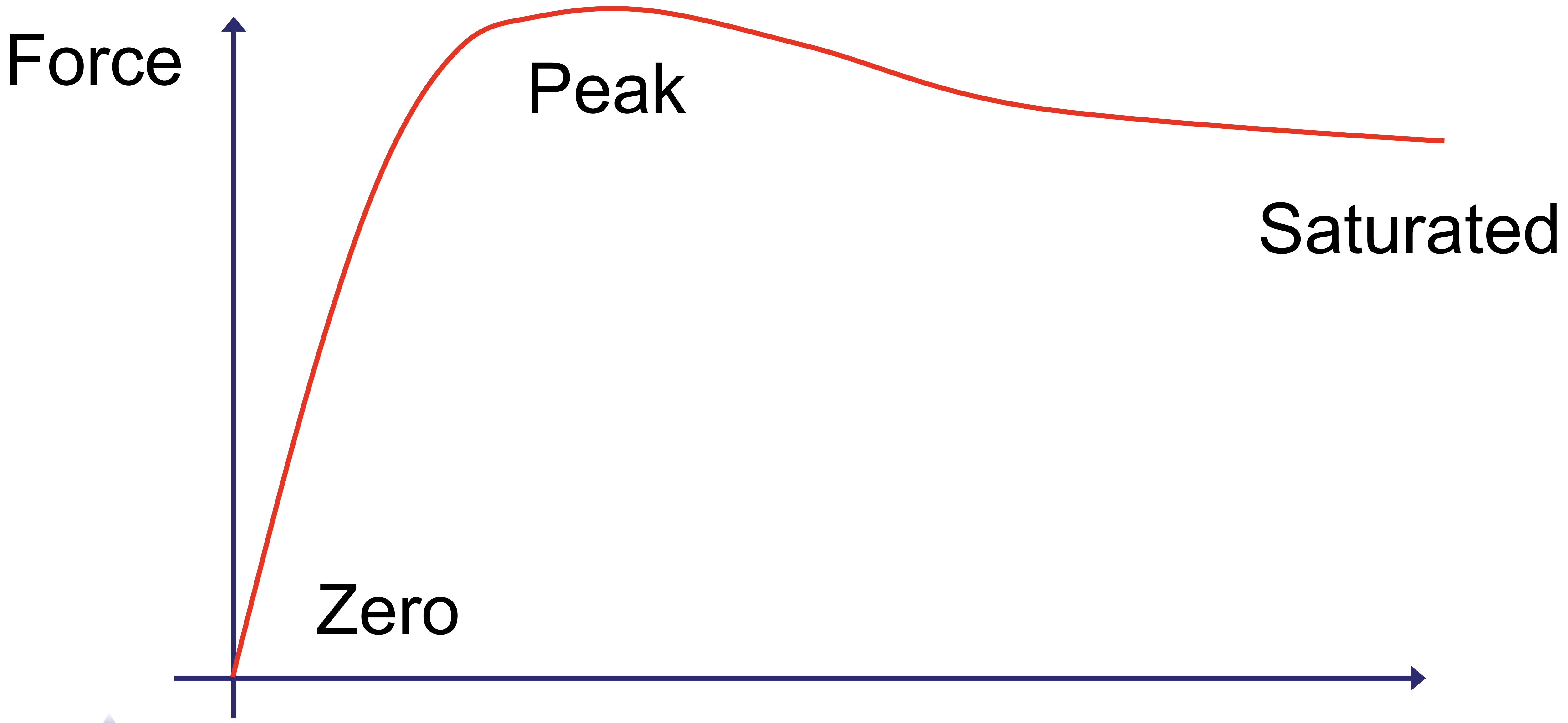


Friction clamp off

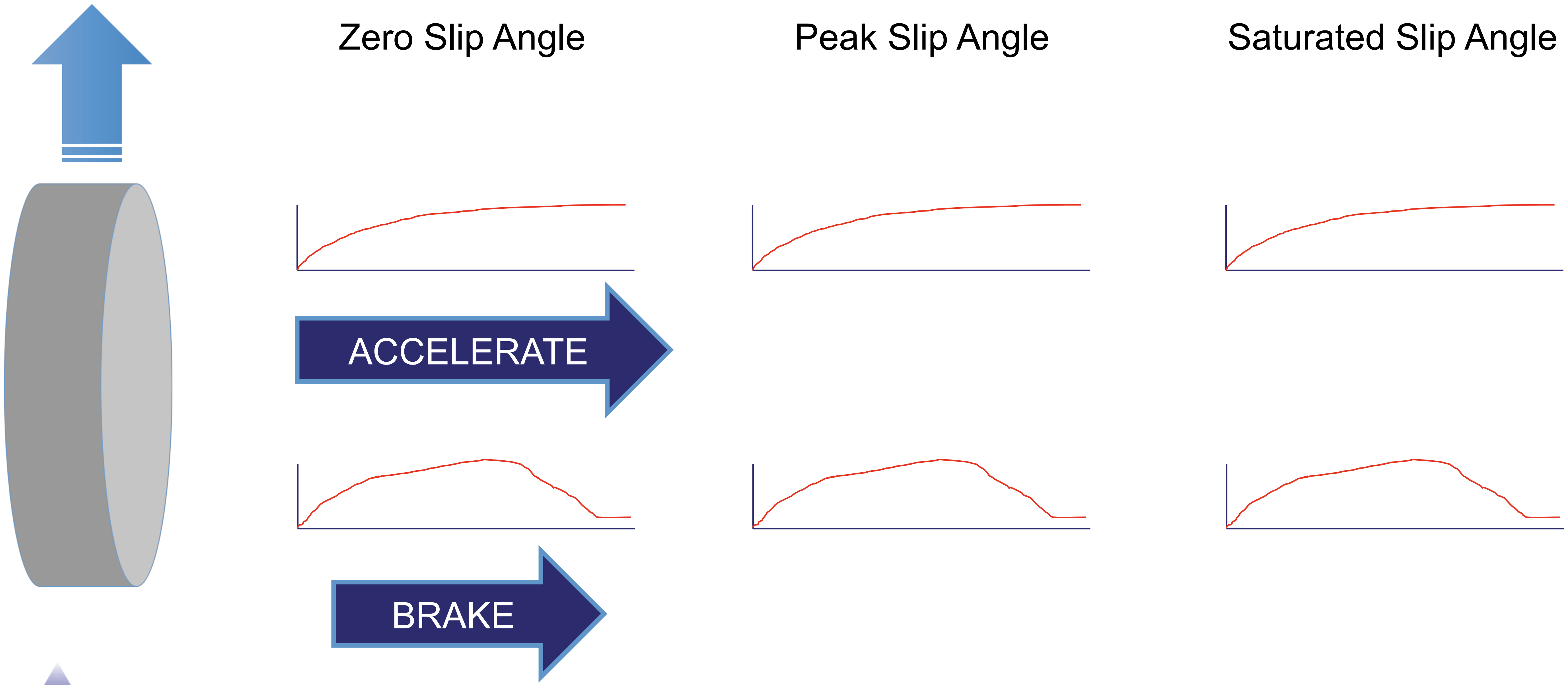
Draw friction curves



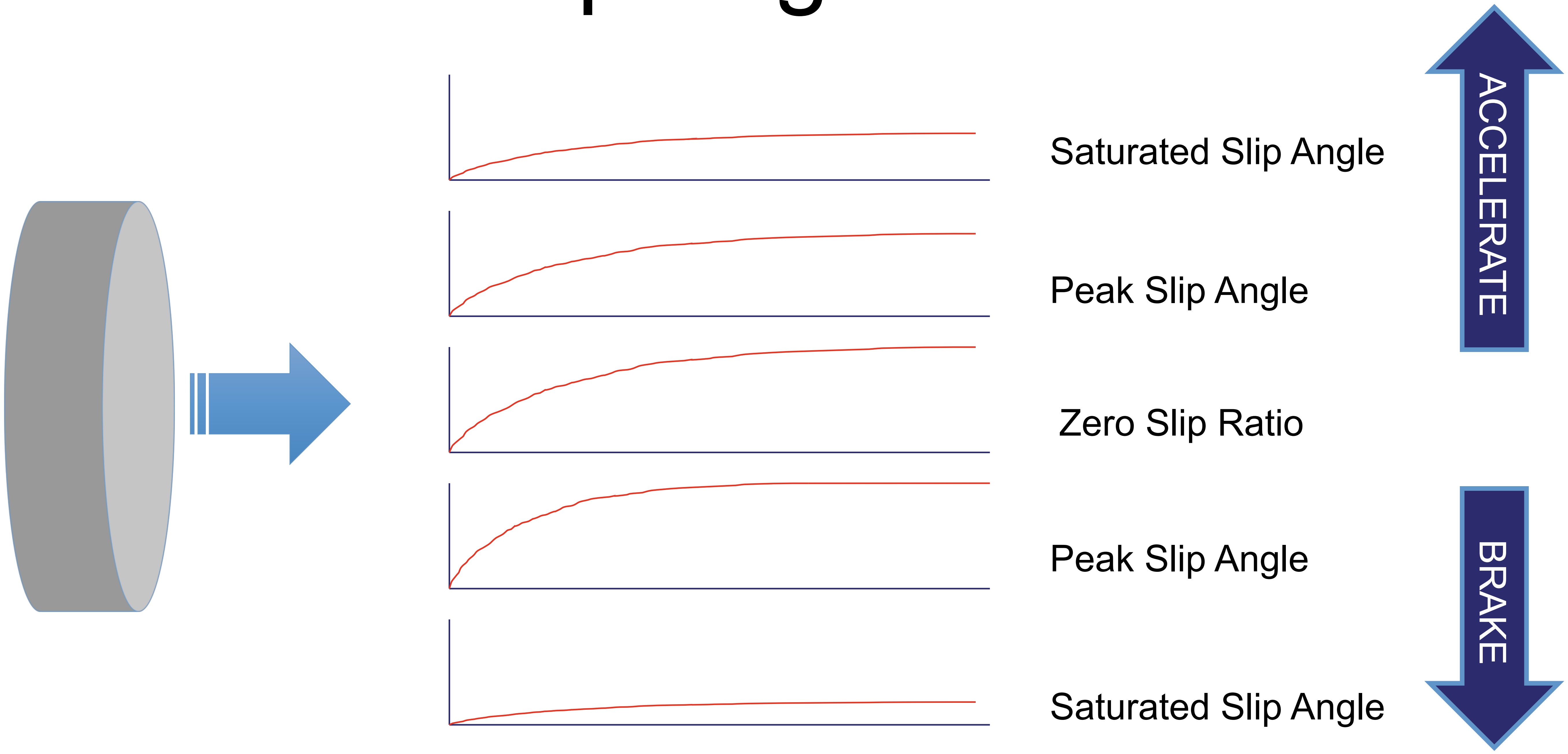
3 phases per direction



Longitudinal force vs Slip Ratio



Lateral force vs Slip Angle



Apply forces

- Grip
  - Authored value usually different for front & rear wheels
  - Material multiplier

$$Impulse = Grip * Graph * WheelLoad * TimeStep$$

Too much grip!

- Reduce pitch and roll angular components
  - Decompose impulse at point to linear impulse & angular
  - Apply factor to roll and pitch components
  - Apply linear impulse and angular to rigid body

Summary

- Similar input parameters as simulation models
- Higher grip than real (especially in braking phase)
- Friction clamps to stay physically stable
- Drawn friction curves
- Scale down pitch and roll components
- Add "driver assists" e.g. drift control on whole vehicle

Drift

- Big topic
- Much like a Kart Racer
- Control the turn of the velocity more directly
- Something for another talk


THANKS!
Q&A
Hamish Young, Lead Mechanics Designer, Avalanche Studios