Technical Design Notes:

By necessity, other than the obvious control of rotating engines to turn and otherwise move the vehicle itself, we need to be off the ground. However, what if the ground isn't a consistent or flat surface, what if it's rounded in some way?

Plan 1

We draw a vector straight down from the vehicle body (center of mass, Y-axis)

This ray will be gravity, all gravity will be local to vehicles.

Plan 2

An elaboration on plan 1, we cast the ray down the vehicle's local Y-axis

We then hit a ground object, or other form of collider that our ray detects.

The specific face that we hit, gravity will be parallel to the vector which is INVERTED from that face's perpendicular normal.

However, we may run into the issue of sliding on convex ground surfaces, or have other issues that affect lateral movement across multiple faces of the track with faces and normals at different angles relative to the world.

Because of this potential problem in particular, we may also need to apply a small torque based on the initial collision angle detected by the first ray we cast to gently push the vehicle to a position above the appropriate normal.

This issue is caused or can be affected by how far above the ground the vehicle is, but I don't think that the overall effect will be significant. Further testing will determine this.

Particle Effect Notes:

Axis Value 0 → 1

Min Velocity → Max Velocity (Initial)
Spread → Focused
Lifetime Longer → Lifetime Shorter (Possibly)

Velocity Min → Max (Could also be tied to axis value)

Orange → Blue (Final Color)

Lifetime 0 → 1

Full Alpha → 0 Alpha (Not necessarily Linear, needs more points)
Size small -> Large -> Small
White → Final Color (See above, either orange or blue)

Additionally,

Could use some sort of slowing or drag effect when hitting ground so it *piles* up.

For Above, we'll get there when we get there. Let's get movement specifics in first.

Cosmetic movement effects.

So, somewhat decidedly, we could or should be moving the object's root node, as far as actual map and game-world traversal goes. This means we can apply cosmetic movement effects to the body (the parent of engines and particle systems) without affecting gameplay in any significant way (if we choose).

Tilt Left and Right

Can tilt the vehicle to either side depending on the ratio of each side's vertical thrust component. Math below.

Simple [Sin(Engine Θ) * (Thrust or Trigger Axis Value)] provides part of what we're checking in our ratio. The resulting value will be between 0 and 1, which we will call EngineValue.

EngineValue(Left) - EngineValue(Right) results in a value between -1 and 1, providing a float value that we can scale a maximum rotation by. In general, this rotation should be small.

If the rotation ends up being inverted/opposite, We do Right - Left instead. The resulting rotation is [RotationValue = MaxRotation * (evLeft - evRight)].

Tilt Forward and Back

This is possibly a little trickier, it would be grand if we could have the nose of the vehicle go down when sharply deccelerating and the nose raise when accelerating.

Current To-do List:

Height raise/lower on the body object isn't taking into account thrust value at all. Additionally, it needs to be refactored to be a minimum value and an additive range. We can also take into account the relative rotation of the body itself now that we have another rotation in our game. Sin of the Roll(X) of the body should work out well enough to scale what our previous value would have been by.

Tentatively disable body roll. It's too jerky, until proper lerping is implemented or chosen.

Notes on Interpolation

The vInterp seemed to work well, or at least I think it was vInterp. It could have been fInterp. Notably, while this wasn't an issue, supposedly rInterp runs into problems as you saw earlier in your development, even though you attempted to constrain the objects' axis of rotation. However, in this particular case, given that fInterp simply needs a delta time (which can be gotten from world value), a speed float which can be modified and the current and target value to interpolate between, there is a strong possibility we can just interpolate the axis input directly and smooth it out that way without messing with rotation.

As the float we get is used in the creation of the rotator(s) to begin with. If that works as is, simply interpolating the float from the axis, we don't need to fuss around with different stages, degrees, or presets of engine rotation and choose when to or when not to interpolate to a new rotation.