# Steering Documentation

# Steering Controller Formulas

Below is a mathematical breakdown of the **steering** portion of the racing algorithm, as reflected in the Unity C# code.

We denote:

- $\{\mathbf{p}_i\}$: **list of checkpoint positions** in world coordinates
- $v$: **current speed** of the car (scalar)
- $\mathbf{pos}$: **current position** of the car (for the geometric calculations)
- $\mathbf{fwd}$: **car's forward direction** (2D projection)
- $\theta$: **signed angle** (in degrees) between $\mathbf{fwd}$ and the direction to the checkpoint
- $d$: **distance** to the selected checkpoint
- $\Delta t$ : **current frame delta time** (i.e., `Time.deltaTime` in Unity)
- $r_{\text{time}}$ : **reaction time**
- $\dot{\theta}$ : **angle-change rate**
- $\theta_{\text{updated}}$ : **updated angle** after applying (\dot{\theta}) over (\Delta t)
- $\alpha' \in [-1, 1]$ : **raw steering input** (before final validation)
- $\alpha$ : **final steering angle** in **radians** after validation

# 1. Lookahead Index

The algorithm first determines how far ahead to look in the checkpoint list, based on the current speed. In code:

$$\ell_{\text{steering}} = \min\Big(\lfloor \text{steeringLookAheadSensitivity} \cdot v \rfloor + 1,\ \text{numberOfCheckpoints} - 1\Big).$$

# 2. Direction and Distance to the Chosen Checkpoint

Let:

$$\mathbf{dir} = \mathbf{p}_{\ell_{\text{steering}}} - \mathbf{pos}, \quad d = \|\mathbf{dir}\|.$$

We then convert $\mathbf{dir}$ and $\mathbf{fwd}$ to 2D vectors (X-Z plane) and compute a signed angle $\theta$:

$$\theta = \text{SignedAngle}\big(\mathbf{fwd},\ \mathbf{dir}\big).$$

> By convention in Unity, `SignedAngle` returns degrees in the range $(-180, 180]$.

# 3. Reaction Time

The “reaction time” depends on distance and speed:

$$r_{\text{time}} = \frac{d}{v}.$$

In the code, if $r_{\text{time}}$ is extremely small (or if $v \approx 0$), the algorithm forces a minimum:

$$\text{if } r_{\text{time}} \approx 0, \quad r_{\text{time}} = 0.1.$$

# 4. Angle Change Rate

The algorithm aims to get $\theta$ to zero in time $r_{\text{time}}$ . Thus:

$$\dot{\theta} = \frac{-\theta}{r_{\text{time}}}.$$

Negative sign ensures we rotate “toward” zero angle.

# 5. Updated Angle

During each frame, we update the angle by:

$$\theta_{\text{updated}} = \theta \; + \; \dot{\theta}\,\Delta t = \theta \; + \; \left(\frac{-\theta}{r_{\text{time}}}\right)\Delta t.$$

# 6. Raw Steering Input

The code then maps $\theta_{\text{updated}}$ onto a steering “fraction” $\alpha'$ in ([-1,1]), since the maximum steering angle is $21^\circ$. Specifically:

$$\alpha' = \text{clamp}\Big(\frac{\theta_{\text{updated}}}{21^\circ}, \; -1, \; 1\Big).$$

The final return from `CalculateSteeringInput` in the code is $-\alpha'$:

$$(\text{raw steering output}) = -\,\alpha'.$$

> The minus sign in the code make sure it aligns with a specific rotation direction desired by the car controller.

# 7. Validate Steering Angle

## 7.1 Scaling to Radians

In Unity’s `ValidateSteeringAngle`, the code converts the ([-1,1]) steering input back into a rotation in radians:

1. **Maximum steering angle in radians** is $21^\circ \approx 0.3665191$ rad.
2. Therefore:

$$\alpha_{\text{rad}} = (0.3665191) \times (\text{raw steering output}).$$

3. The code then returns $-\alpha_{\text{rad}}$. So the final steering angle $\alpha$ (in radians) is:

$$\alpha = -\,\alpha_{\text{rad}} = -\,\big(0.3665191\ \times\ (-\,\alpha')\big) = 0.3665191\,\alpha'.$$

   (Or if you track each sign precisely: The code in `ValidateSteeringAngle` returns `–maxSteeringAngleRadians`, and we already had a negative from `CalculateSteeringInput`. In practice, the net effect is just ensuring the correct sign for steering in the physics simulation.)

---

# Complete Steering Function

Let us combine all steps into a single function,

$$S_c\big(\{\mathbf{p}_i\},\, v\big),$$

where the output is the **final steering angle** in radians (as the simulation applies it). Note that we require the car's position $\mathbf{pos}$ and forward direction $\mathbf{fwd}$ for geometric calculations, even though we only list $\{\mathbf{p}_i\}$ and $v$ as main inputs.

$$
\begin{aligned}
&1.\ \ell = \min\Big(\lfloor \text{steeringLookAheadSensitivity} \cdot v \rfloor + 1,\ \#\text{checkpoints} - 1\Big),\\
&2.\ \mathbf{dir} = \mathbf{p}_\ell - \mathbf{pos}, \quad d = \|\mathbf{dir}\|,\\
&3.\ \theta = \text{SignedAngle}(\mathbf{fwd},\ \mathbf{dir}),\\
&4.\ r_{\text{time}} = \max\Big(\tfrac{d}{v},\ 0.1\Big),\\
&5.\ \dot{\theta} = -\frac{\theta}{r_{\text{time}}},\\
&6.\ \theta_{\text{updated}} = \theta + \dot{\theta}\,\Delta t,\\
&7.\ \alpha' = \text{clamp}\Big(\tfrac{\theta_{\text{updated}}}{21^\circ},\ -1,\ 1\Big),\\
&\quad\ (\text{raw steering input in } [-1, 1])\\
&8.\ \text{raw steering out} = -\alpha',\\
&9.\ \alpha_{\text{rad}} = 0.3665191 \times \text{raw steering out},\\
&10.\ \alpha = -\alpha_{\text{rad}}.\\
&\quad\ \boxed{S_c\big(\{\mathbf{p}_i\},\, v\big) = \alpha.}
\end{aligned}
$$

Hence, $S_c$ yields the final steering angle $\alpha$ in radians, suitable for the simulation.

---

# Important Relations

1. **Lookahead**
   - Similar to throttle, the code chooses how far ahead $(\ell)$ based on speed.
2. **Angle-to-Zero Strategy**
   - The negative of $\theta / r_{\text{time}}$ ensures the car tries to steer so that angle eventually goes to zero within $r_{\text{time}}$.
3. **Max Steering Angle**
   - The code uses $21^\circ$ as the maximum feasible wheel angle.
4. **Sign Conventions**
   - There are multiple sign inversions (both in `CalculateSteeringInput` and `ValidateSteeringAngle`) to align with the final desired orientation for the car's physics.
5. **Radians vs. Degrees**
   - Internal logic for steering uses degrees for the angle ratio, then final "validated" steering angle is expressed in radians.