**Where does the Reward come from?**

Reinforcement learning algorithms are geared for optimization of cumulative rewards.

The model will learn which action (and then subsequent actions) will result in the highest cumulative reward on the way to the goal.

The critical part to make your reinforcement learning model work is the *reward function*. In general you design your reward function to act like an incentive plan.

**Reward function parameters for AWS DeepRacer**

In AWS DeepRacer, the reward function is a Python function which is given certain parameters that describe the current state and returns a numeric reward value.

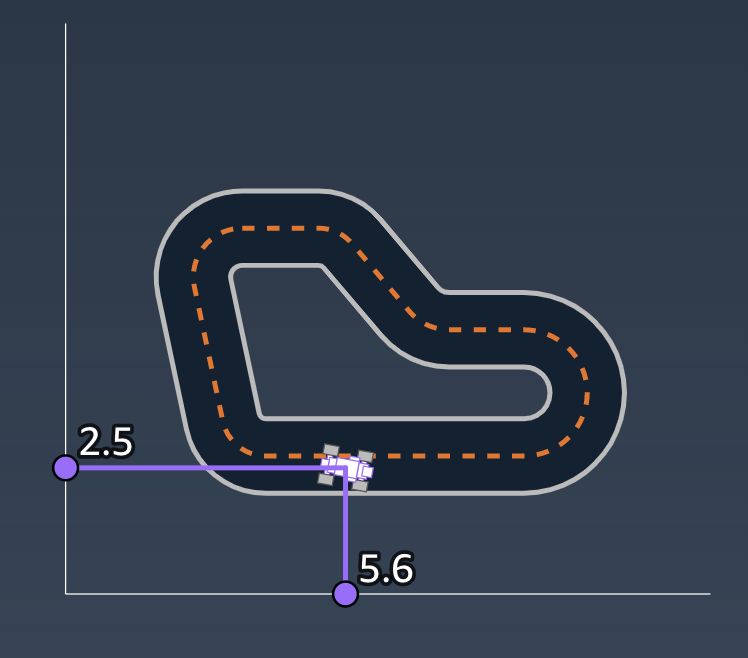
The parameters passed to the reward function describe various aspects of the state of the vehicle, such as its position and orientation on the track, its observed speed, steering angle and more.

We will explore some of these parameters and how they describe the vehicle as it drives around the track:

* Position on track
* Heading
* Waypoints
* Track width
* Distance from center line
* All wheels on track
* Speed
* Steering angle

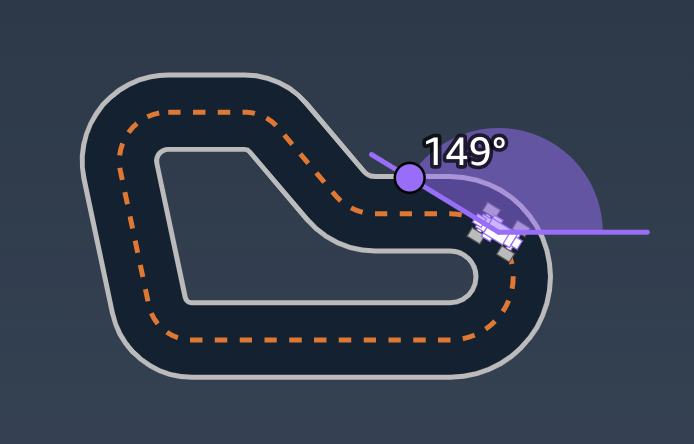
**1. Position on track**

The parameters x and y describe the position of the vehicle in meters, measured from the lower-left corner of the environment.



**2. Heading**

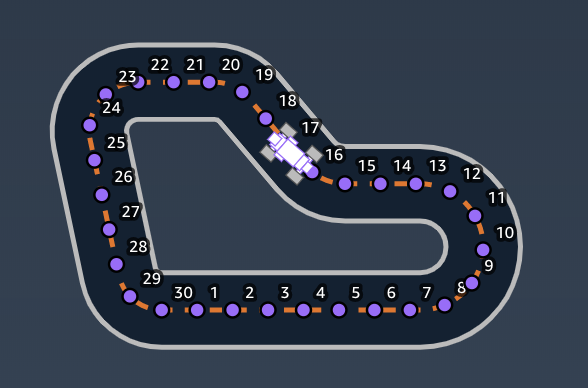
The heading parameter describes the orientation of the vehicle in degrees, measured counter-clockwise from the X-axis of the coordinate system.



**3. Waypoints**

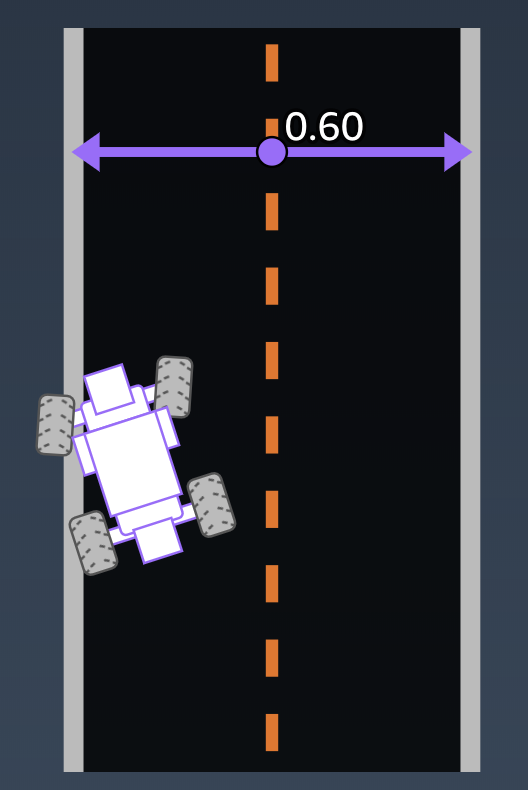
The waypoints parameter is an ordered list of milestones placed along the track center.

Each waypoint in waypoints is a pair [x, y] of coordinates in meters, measured in the same coordinate system as the car's position.



**4. Track width**

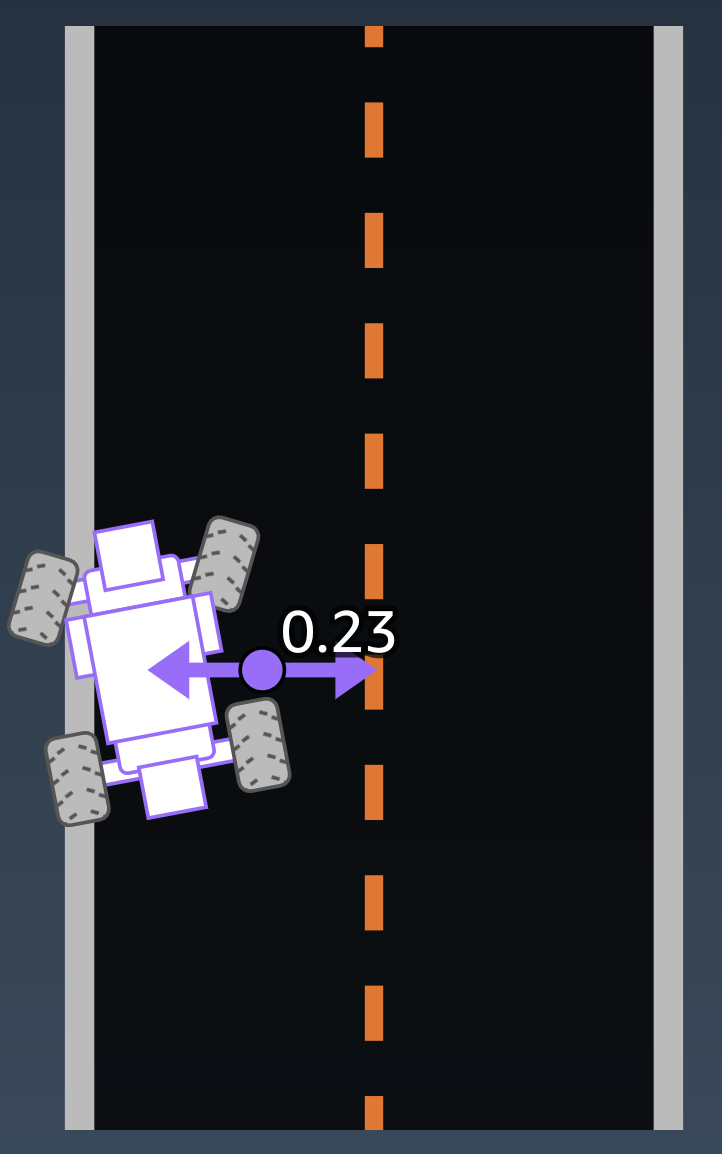
The track\_width parameter is the width of the track in meters.



**5. Distance from center line**

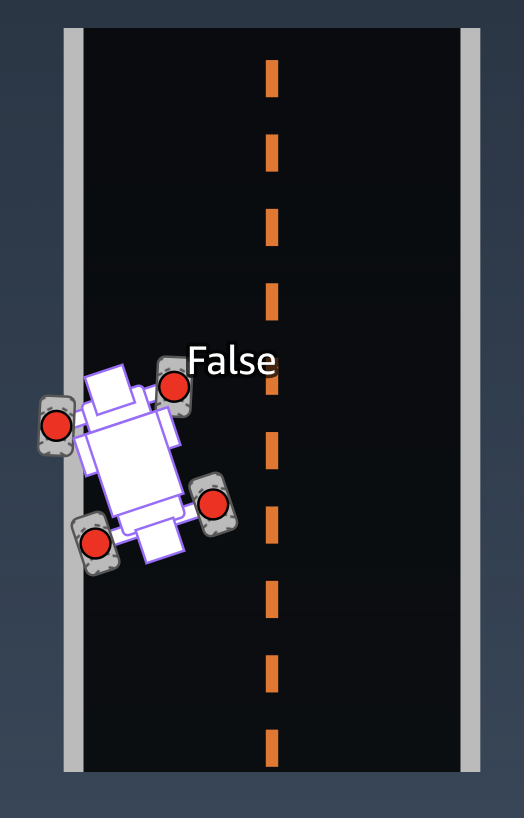
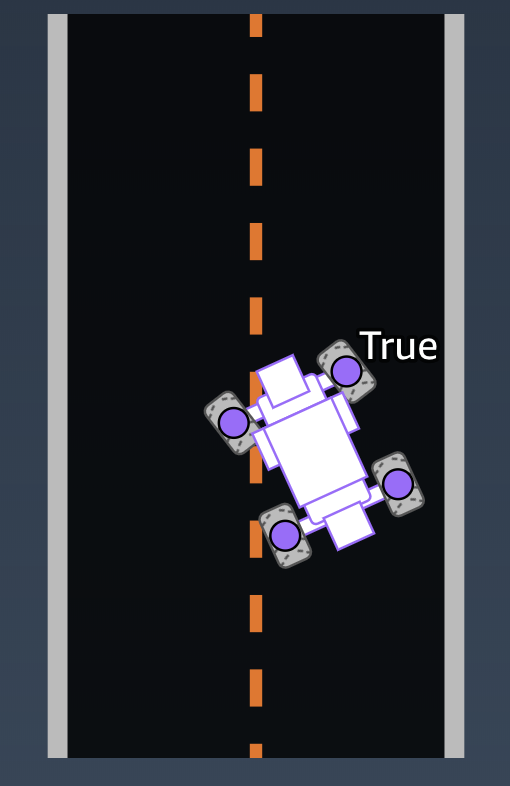
The distance\_from\_center parameter measures the displacement of the vehicle from the center of the track.

The is\_left\_of\_center parameter is a boolean describing whether the vehicle is to the left of the center line of the track.



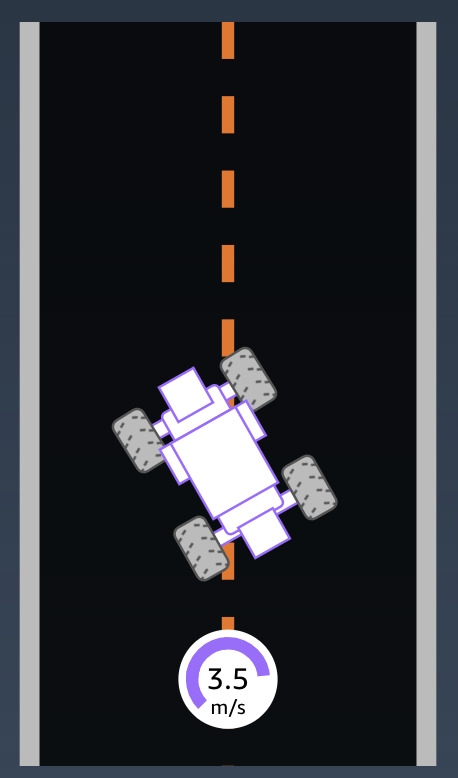
**6. All wheels on track**

The *all\_wheels\_on\_track* parameter is a boolean (true / false) which is true if all four wheels of the vehicle are inside the track borders, and false if any wheel is outside the track.

**7. Speed**

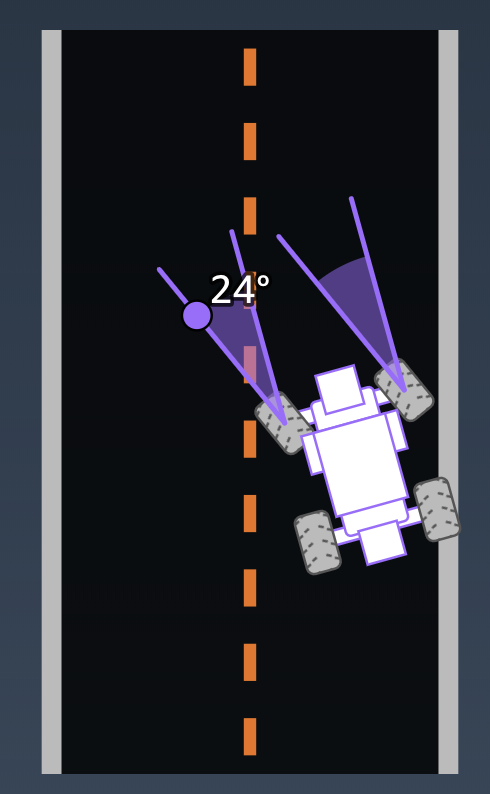
The speed parameter measures the observed speed of the vehicle, measured in meters per second.



**8. Steering angle**

The steering\_angle parameter measures the steering angle of the vehicle, measured in degrees.

This value is negative if the vehicle is steering right, and positive if the vehicle is steering left.



**Summary**

In total there are 13 parameters you can use in your reward function

|  |  |
| --- | --- |
| *x and y* | The position of the vehicle on the track |
| *heading* | Orientation of the vehicle on the track |
| *waypoints* | List of waypoint coordinates |
| *closest\_waypoints* | Index of the two closest waypoints to the vehicle |
| *progress* | Percentage of track completed |
| *steps* | Number of steps completed |
| *track\_width* | Width of the track |
| *distance\_from\_center* | Distance from track center line |
| *is\_left\_of\_center* | Whether the vehicle is to the left of the center line |
| *all\_wheels\_on\_track* | Is the vehicle completely within the track boundary? |
| *speed* | Observed speed of the vehicle |
| *steering\_angle* | Steering angle of the front wheels |

For more information on these parameters and the values they can take, [read the detailed documentation](https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-console-train-evaluate-models.html#deepracer-reward-function-signature).

**Putting it all together**

With all these parameters at your disposal, you can define a reward function to incentivize whatever driving behavior you like.

Let's see a few examples of reward functions and how they use the parameters to determine a reward. The following three reward functions are available as examples in the AWS DeepRacer console so you can try them out and see how they behave, or submit them to the AWS DeepRacer League.

# The Reward Function

**1. Stay On Track**

In this example, we give a high reward for when the car stays on the track, and penalize if the car deviates from the track boundaries.

This example uses the all\_wheels\_on\_track, distance\_from\_center and track\_width parameters to determine whether the car is on the track, and give a high reward if so.

Since this function doesn't reward any specific kind of behavior besides staying on the track, an agent trained with this function may take a longer time to converge to any particular behavior.

|  |
| --- |
| def reward\_function(params):  '''  Example of rewarding the agent to stay inside the two borders of the track  '''  # Read input parameters  all\_wheels\_on\_track = params['all\_wheels\_on\_track']  distance\_from\_center = params['distance\_from\_center']  track\_width = params['track\_width']  # Give a very low reward by default  reward = 1e-3  # Give a high reward if no wheels go off the track and  # the agent is somewhere in between the track borders  if all\_wheels\_on\_track and (0.5\*track\_width - distance\_from\_center) >= 0.05:  reward = 1.0  # Always return a float value  return float(reward) |

**2. Follow Center Line**

In this example we measure how far away the car is from the center of the track, and give a higher reward if the car is close to the center line.

This example uses the track\_width and distance\_from\_center parameters, and returns a decreasing reward the further the car is from the center of the track.

This example is more specific about what kind of driving behavior to reward, so an agent trained with this function is likely to learn to follow the track very well. However, it is unlikely to learn any other behavior such as accelerating or braking for corners.

|  |
| --- |
| def reward\_function(params):  '''  Example of rewarding the agent to follow center line  '''  # Read input parameters  track\_width = params['track\_width']  distance\_from\_center = params['distance\_from\_center']  # Calculate 3 markers that are at varying distances away from the center line  marker\_1 = 0.1 \* track\_width  marker\_2 = 0.25 \* track\_width  marker\_3 = 0.5 \* track\_width  # Give higher reward if the car is closer to center line and vice versa  if distance\_from\_center <= marker\_1:  reward = 1.0  elif distance\_from\_center <= marker\_2:  reward = 0.5  elif distance\_from\_center <= marker\_3:  reward = 0.1  else:  reward = 1e-3 # likely crashed/ close to off track  return float(reward) |

**3. No incentive**

An alternative strategy is to give a constant reward on each step, regardless of how the car is driving.

This example doesn't use any of the input parameters — instead it returns a constant reward of 1.0 on each step.

The agent's only incentive is to successfully finish the track, and it has no incentive to drive faster or follow any particular path. It may behave erratically.

However, since the reward function doesn't constrain the agent's behavior, it may be able to explore unexpected strategies and behaviors that turn out to perform well.

|  |
| --- |
| def reward\_function(params):  '''  Example of no incentive  '''  # Always return 1 if the car does not crash  return 1.0 |

**The reward function**

In reinforcement learning, the reward function is the primary code used to incentivize optimal actions. It’s a mechanism used by the environment to let the agent know how it’s doing. The agent takes a particular action in a given state and receives either an immediate reward or a penalty.

For AWS DeepRacer, the reward function is vital to optimizing the models and enhancing performance around the track. For instance, when using the AWS DeepRacer console to train a model with a supported framework, the reward function is the only application-specific piece, and it depends on your input.

AWS DeepRacer includes various parameters that will help you optimize your reinforcement learning algorithm. Let’s take a look at those below before diving into our next video. Click on each parameter to learn more.

**all\_wheels\_on\_track**

–

Type: Boolean

Syntax: params['all\_wheels\_on\_track']

Description: If all four wheels are on the track, where *track*is defined as the road surface including the border lines, then all\_wheels\_on\_track is True. If any wheel is off the track, then all\_wheels\_on\_track is False. Note: If all four wheels are off the track, the car will be reset.

**x**

–

Type: Float

Syntax: params['x']

Description: Returns the x coordinate of the center of the front axle of the car, in unit meters.

**y**

–

Type: Float

Syntax: params['y']

Description: Returns the y coordinate of the center of the front axle of the car, in unit meters.

**distance\_from\_center**

–

Type: Float [0, track\_width/2]

Syntax: params['distance\_from\_center']

Description: Absolute distance from the center of the track. The center of the track is determined by the line that links all center waypoints.

**is\_left\_of\_center**

–

Type: Boolean

Syntax: params['is\_left\_of\_center']

Description: A variable that indicates if the car is to the left of the track's center.

**is\_reversed**

–

Type: Boolean

Syntax: params['is\_reversed']

Description: A variable that indicates whether the car is training in the original direction of the track or the reverse direction of the track.

**heading**

–

Type: Float (-180,180]

Syntax: params['heading']

Description: Returns the heading that the car is facing in degrees. When the car faces the direction of the x-axis increasing (with y constant), then it will return 0. When the car faces the direction of the y-axis increasing (with x constant), then it will return 90. When the car faces the direction of the y-axis decreasing (with x constant), then it will return -90.

**progress**

–

Type: Float [0,100]

Syntax: params['progress']

Description: Percentage of the track complete. A progress of 100 indicates that the lap is completed.

**steps**

–

Type: Integer

Syntax: params['steps']

Description: Number of steps completed. One step is one state, action, next state, reward tuple.

**speed**

–

Type: Float

Syntax: params['speed']

Description: The desired speed of the car in meters per second. This should match the selected action space. In other words, define this parameter within the limit that you set in the action space.

**steering\_angle**

–

Type: Float

Syntax: params['steering\_angle']

Description: The desired steering angle of the car in degrees. This should match the selected action space. (In other words, define this parameter within the limit that you set in the action space. Note that positive angles (+) indicate going left, and negative angles (-) indicate going right. This is aligned with 2D geometric processing.

**track\_width**

–

Type: Float

Syntax: params['track\_width']

Description: The width of the track, in unit meters.

**waypoints**

–

Type: List

Syntax: params['waypoints'] for the full list or params['waypoints'][i] for the i-th waypoint

Description: Ordered list of waypoints that are spread around the center of the track, with each item in the list being the (x, y) coordinate of the waypoint. The list starts at zero.

**closest\_waypoints**

–

Type: Integer

Syntax: params['closest\_waypoints'][0] or params['closest\_waypoints'][1]

Description: Returns a list containing the nearest previous waypoint index, and the nearest next waypoint index. params['closest\_waypoints'][0] returns the nearest previous waypoint index, and params['closest\_waypoints'][1] returns the nearest next waypoint index.

For an illustration of some of the parameters listed above, see the image below.

