
AWS DeepRacer

Developer Guide



AWS DeepRacer: Developer Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is AWS DeepRacer?	1
The AWS DeepRacer Console	1
The AWS DeepRacer Vehicle	1
The DeepRacer League	2
An Integrated Learning System	2
Basic Concepts and Terminology	2
Get Started	6
Set Up	6
Set Up AWS Account	6
Create an IAM User	7
AWS DeepRacer-Dependent Services	7
Required IAM Roles	9
Train Your First Model	9
Evaluate Models in Simulation	14
Train and Evaluate Models	20
Understand Reinforcement Learning	21
Overview of Reinforcement Learning	21
Action Space and Reward Function	23
Training Algorithm	24
AWS DeepRacer Service Architecture	26
AWS DeepRacer Workflow	27
Simulated-to-Real Performance Gaps	28
Train and Evaluate Models Using AWS DeepRacer Console	28
Create Your Reward Function	29
Explore Action Space	33
Tune Hyperparameters	34
Clone a Trained Model	37
Evaluate Models in Simulations	38
Log Events to CloudWatch Logs	38
Optimize Training for Real Environments	40
Train and Evaluate Models Using Amazon SageMaker Notebooks	42
Create a Notebook	42
Initialize the Notebook Instance	43
Set Up the Environment	47
Train Your AWS DeepRacer Model	50
Reward Function Input Parameters	57
all_wheels_on_track	58
closest_waypoints	60
distance_from_center	61
heading	63
is_left_of_center	63
progress	63
speed	64
steering_angle	64
steps	66
track_width	66
x, y	68
waypoints	68
Operate Your Vehicle	70
Prep Your Vehicle	70
Inspect Your Vehicle	71
Charge Your Vehicle	72
Test Compute Module	73
Turn Off Your Device	74

Vehicle Chassis Parts	74
Set Up Your Vehicle	81
Get Ready for Choose a Wi-Fi	82
Set Up Wi-Fi and Update Software	82
Sign In to DeepRacer Device Console	83
Calibrate Your Vehicle	85
Upload Your Model	91
Drive Your Vehicle	92
Drive Your AWS DeepRacer Vehicle Manually	92
Drive Your AWS DeepRacer Vehicle Autonomously	93
Manage Vehicle Settings	94
View Vehicle Logs	98
Build Your Physical Track	100
Materials and Tools	100
Materials You May Need	100
Tools You May Need	100
Lay Your Track	101
Dimensional Requirements	101
Model Performance Considerations	102
Steps to Build the Track	102
Track Design Templates	105
Single-Turn Track Template	105
S-Curve Track Template	106
Loop Track Template	106
re:Invent 2018 Track Template	107
Rank Models in Leaderboard	109
Submit Your Model to a Leaderboard	109
Troubleshooting	115
How to Switch AWS DeepRacer Compute Module Power Source from Battery to Power Outlet?	115
How to Connect Your AWS DeepRacer to Your Wi-Fi Network?	117
How to Charge the AWS DeepRacer Drive Module Battery?	117
How to Charge the AWS DeepRacer Compute Module Battery?	118
How to Maintain Vehicle Connection?	119
How to Troubleshoot Wi-Fi Connection if Vehicle's Wi-Fi LED Indicator Flashes Blue, Then Turns Red for Two Seconds, and Finally Off?	119
What Does It Mean When the Vehicle's Wi-Fi or Power LED Indicator Flashes Blue?	119
How Can I Connect to Vehicle's Device Console Using its Hostname?	120
How to Connect to Vehicle's Device Console Using its IP Address?	120
How to Get Device's Mac Address?	120
How to Recover Device Controller Default Password?	121
How to Manually Update Your Device?	122
How to Restore Vehicle to Factory Settings?	123
Prepare for Factory Reset	123
Restore to Factory Settings	136
Document History	137
AWS Glossary	138

What Is AWS DeepRacer?

AWS DeepRacer is an integrated learning system for users of all levels to learn and explore [reinforcement learning \(p. 3\)](#) and to experiment and build autonomous driving applications. It consists of the following components:

- An [AWS Machine Learning](#) service to [train and evaluate reinforcement learning models \(p. 20\)](#) in a [simulated autonomous-driving simulator](#).
- AWS DeepRacer 1/18th scale RC vehicles capable of [running inference on a trained AWS DeepRacer model \(p. 70\)](#) for autonomous driving.
- DeepRacer League for the participants to compete on model performances on a specific track in a simulated or real environment and have the [results ranked on a virtual or physical leaderboard \(p. 109\)](#).

Topics

- [The AWS DeepRacer Console \(p. 1\)](#)
- [The AWS DeepRacer Vehicle \(p. 1\)](#)
- [The AWS DeepRacer League \(p. 2\)](#)
- [AWS DeepRacer As an Integrated Learning System \(p. 2\)](#)
- [AWS DeepRacer Basic Concepts and Terminology \(p. 2\)](#)

The AWS DeepRacer Console

The AWS DeepRacer console is a graphical user interface to interact with the AWS DeepRacer service. You can use the console to train a reinforcement learning model and to evaluate the model performance in the AWS DeepRacer simulator built upon AWS RoboMaker. In the console, you can also download a trained model for deployment to your AWS DeepRacer vehicle for autonomous driving in a physical environment.

In summary, the AWS DeepRacer console supports the following features:

- Create a training job to train a reinforcement learning model with a specified reward function, optimization algorithm, environment, and hyperparameters.
- Choose a simulated track to train and evaluate a model by using Amazon SageMaker and AWS RoboMaker.
- Clone a trained model to improve training by tuning hyperparameters to optimize your model's performance.
- Download a trained model for deployment to your AWS DeepRacer vehicle so it can drive in a physical environment.
- Submit your model to a virtual race and have its performance ranked against other models in a virtual leaderboard.

The AWS DeepRacer Vehicle

The AWS DeepRacer vehicle is a Wi-Fi enabled, physical vehicle that can drive itself on a physical track by using a reinforcement learning model.

- You can manually control the vehicle, or deploy a model for the vehicle to drive autonomously.

- The autonomous mode runs inference on the vehicle's compute module. Inference uses images that are captured from the camera that is mounted on the front.
- A Wi-Fi connection allows the vehicle to download software. The connection also allows the user to access the device console to operate the vehicle by using a computer or mobile device.

The AWS DeepRacer League

The DeepRacer League is an important component of AWS DeepRacer. The DeepRacer League is intended to foster communal learning and collaborative exploration through sharing and competition.

With the DeepRacer League, you can have your development effort compared with other AWS DeepRacer developers in a physical or virtual racing event. Not only do you get a chance to win prizes, you also have a way to measure your reinforcement learning model. You can create opportunities to share your insights with other participants, to learn from each other, and to inspire each other.

AWS DeepRacer As an Integrated Learning System

Reinforcement learning, especially deep reinforcement learning, has proven effective in solving a wide array of autonomous decision-making problems. It has applications in financial trading, data center cooling, fleet logistics, and autonomous racing, to name a few.

Reinforcement learning has the potential to solve real-world problems. However, it has a steep learning curve because of the extensive technological scope and depth. Real-world experimentation requires that you construct a physical agent, e.g., an autonomous racing car. It also requires that you secure a physical environment, e.g., a driving track or public road. The environment can be costly, hazardous, and time-consuming. These requirements go beyond merely understanding reinforcement learning.

To help reduce the learning curve, AWS DeepRacer simplifies the process in three ways:

- By offering a wizard to guide training and evaluating reinforcement learning models. The wizard includes pre-defined environments, states, actions, and customizable reward functions.
- By providing a simulator to emulate interactions between a virtual [agent \(p. 4\)](#) and a virtual environment.
- By offering an AWS DeepRacer vehicle as a physical agent. Use the vehicle to evaluate a trained model in a physical environment. This closely resembles a real-world use case.

If you are a seasoned machine learning practitioner, you will find AWS DeepRacer a welcome opportunity to build reinforcement learning models for autonomous racing in both virtual and physical environments. To summarize, use AWS DeepRacer to create reinforcement learning models for autonomous racing with the following steps:

1. Train a custom reinforcement learning model for autonomous racing. Do this by using the AWS DeepRacer console integrated with Amazon SageMaker and AWS RoboMaker.
2. Use the AWS DeepRacer simulator to evaluate a model and test autonomous racing in a virtual environment.
3. Deploy a trained model to AWS DeepRacer model vehicles to test autonomous racing in a physical environment.

AWS DeepRacer Basic Concepts and Terminology

AWS DeepRacer builds on the following concepts and uses the following terminology.

AWS DeepRacer

Also referred to as AWS DeepRacer vehicle. One type of AWS DeepRacer vehicle is an AWS DeepRacer car that is a 1/18th scale model car. It has a mounted camera and an on-board compute module. The compute module runs inference in order to drive itself along a track. The compute module and the vehicle chassis are powered by dedicated batteries known as the compute battery and the drive battery, respectively.

AWS DeepRacer service

An AWS Machine Learning service for exploring reinforcement learning that is focused on autonomous racing. The AWS DeepRacer service supports the following features:

1. Train a reinforcement learning model on the cloud.
2. Evaluate a trained model in the AWS DeepRacer console.
3. Submit a trained model to a virtual race and, if qualified, have its performance posted to the event's leaderboard.
4. Clone a trained model continue training for improved performances.
5. Download the trained model artifacts for uploading to an AWS DeepRacer vehicle.
6. Place the vehicle on a physical track for autonomous driving and evaluate the model for real-world performances.
7. Run the AWS DeepRacer League for racing events on physical tracks.
8. Remove unnecessary charges by deleting models that you don't need.

Reinforcement learning

A machine learning method that is focused on autonomous decision making by an agent in order to achieve specified goals through interactions with an environment. In reinforcement learning, learning is achieved through trial and error and training does not require labeled input. Training relies on the reward hypothesis. The hypothesis is that all goals can be achieved by maximizing a future reward after action sequences. In reinforcement learning, designing the reward function is important. The better the reward function is crafted, the better the agent can decide what actions to take to reach the goal.

For autonomous racing, the agent is a vehicle. The environment includes traveling routes and traffic conditions. The goal is for the vehicle to reach its destination quickly without accidents. Rewards are scores used to encourage safe and speedy travel to the destination. The scores penalize dangerous and wasteful driving.

To encourage learning during training, the learning agent must be allowed to sometimes pursue actions that might not result in rewards. This is referred to as the exploration and exploitation trade-off. It helps reduce or remove the likelihood that the agent might be misguided into false destinations.

For a more formal definition, see [reinforcement learning](#) on Wikipedia.

Reinforcement learning model

The environment in which an agent acts establishes three things: The states that the agent has, the actions that the agent can take, and the rewards that are received by taking action. The strategy with which the agent decides its action is referred to as a policy. The policy takes the environment state as input and outputs the action to take. In reinforcement learning, the policy is often represented by a deep neural network and we refer to this as the reinforcement learning model. Each training job generates one model. A model can be generated even if the training job is stopped early. A model is immutable, which means it cannot be modified and overwritten after it's created.

AWS DeepRacer simulator

A virtual environment built on AWS RoboMaker for visualizing training and evaluating AWS DeepRacer models.

AWS DeepRacer vehicle

See [AWS DeepRacer \(p. 3\)](#)

AWS DeepRacer car

A type of [AWS DeepRacer vehicle \(p. 4\)](#) that is a 1/18th scale model car.

Leaderboard

A *leaderboard* is a ranked list of AWS DeepRacer vehicle performances in an AWS DeepRacer League racing event. The race can be a virtual event, carried out in the simulated environment, or a physical event, carried out in a real-world environment. The performance metric is the average lap time submitted by AWS DeepRacer users who have evaluated their trained models on a track identical or similar to the given track of the race.

If a vehicle completes three laps consecutively, then it qualifies to be ranked on a leaderboard. The average lap time for the first three consecutive laps is submitted to the leaderboard.

Machine-learning frameworks

The software libraries used to build machine learning algorithms. Supported frameworks for AWS DeepRacer include Tensorflow.

Policy network

A policy network is a neural network that is trained. It takes video images as input and predicts the next action for the agent. Depending on the algorithm, it may also evaluate the value of current state of the agent.

Optimization algorithm

An optimization algorithm is the algorithm used to train a model. For supervised training, the algorithm is optimized by minimizing a loss function with a particular strategy to update weights. For reinforcement learning, the algorithm is optimized by maximizing the expected future rewards with a particular reward function.

Neural network

A collection of connected units or nodes that are used to build an information model based on biological systems. Also referred to as artificial neural network. Each node is called an artificial neuron and mimics a biological neuron in that it receives an input (stimulus), becomes activated if the input signal is strong enough (activation), and produces an output predicated upon the input and activation. It's widely used in machine learning because an artificial neural network can serve as a general-purpose approximation to any function. Teaching machine to learn becomes finding the optimal function approximation for the given input and output. In deep reinforcement learning, the neural network represents the policy and is often referred to as the policy network. Training the policy network amounts to iterating through steps that involve generating experiences based on the current policy followed by optimizing the policy network with the newly generated experiences. The process continues until certain performance metrics meet required criteria.

Hyperparameters

Algorithm-dependent variables that control the performance of training a neural network. An example hyperparameter is the learning rate that controls how much new experiences are counted for in learning at each step. A larger learning rate makes a faster training but may make the trained model lower quality. Hyperparameters are empirical and require systematic tuning for each training.

AWS DeepRacer Track

A path or course on which an AWS DeepRacer vehicle drives. The track can exist in either a simulated or real-world, physical environment. You use a simulated environment for training an AWS DeepRacer model on a virtual track. The AWS DeepRacer console makes virtual tracks available. You use a real-world environment for running an AWS DeepRacer vehicle on a physical track. The

DeepRacer League provides physical tracks for event participants to compete. You must create your own physical track if you want to run your AWS DeepRacer vehicle in any other situation.

Reward function

An algorithm within a learning model that tells the agent whether the action performed resulted in:

- A good outcome that should be reinforced.
- A neutral outcome.
- A bad outcome that should be discouraged.

The reward function is a key part of reinforcement learning. It determines the behavior that the agent learns by incentivizing specific actions over others. The user provides the reward function by using Python. This reward function is used by an optimizing algorithm to train the reinforcement learning model.

Experience episode

A period in which the agent collects experiences as training data from the environment by running from a given starting point to completing the track or going off the track. Different episodes can have different lengths. Also referred to as an episode or experience-generating episode.

Experience iteration

Consecutive experiences between each policy iteration that performs updates of the policy network weights. Also referred to as an experience-generating iteration, the size can be set in one of the hyperparameters for training. At the end of each experience iteration, the collected episodes are added to an experience replay or buffer. The neural network is updated by using random samples of the experiences.

Policy iteration

Any number of passes through the randomly sampled training data to update the policy neural network weights during gradient ascent. A single pass through the training data to update the weights is also known as an epoch. Policy iteration is also referred to as policy-updating iteration.

Training job

A workload that trains a reinforcement learning model and creates trained model artifacts on which to run inference. Each training job has two sub-processes:

1. Start the agent to follow the current policy. The agent explores the environment in a number of [episodes \(p. 5\)](#) and creates training data. This data generation is an iterative process itself.
2. Apply the new training data to compute new policy gradients. Update the network weights and continue training. Repeat Step 1 until a stop condition is met.

Each training job produces a trained model and outputs the model artifacts to a specified data store.

Evaluation job

A workload that tests the performance of a model. Performance is measured by given metrics after the training job is done. The standard AWS DeepRacer performance metric is the driving time that an agent takes to complete a lap on a track. Another metric is the percentage of the lap completed.

Get Started with AWS DeepRacer

To help you get started quickly with AWS DeepRacer, we walk you through essential steps using the AWS DeepRacer console. Use the console to create a training job to train your first AWS DeepRacer model for autonomous racing. Use the console to evaluate a trained model in a simulated environment.

Topics

- [Set Up for Using AWS DeepRacer \(p. 6\)](#)
- [Train Your First AWS DeepRacer Model for Autonomous Racing \(p. 9\)](#)
- [Evaluate Your AWS DeepRacer Models in Simulation \(p. 14\)](#)

Set Up for Using AWS DeepRacer

Before you can begin using AWS DeepRacer, sign up for the service with an AWS account and create an IAM user. If you have not already finished these two steps, follow the instructions here.

To run AWS DeepRacer securely, you must have appropriate security settings configured in your AWS account. The AWS DeepRacer console makes it easy to configure these security settings and automatically create the required account resources. They include VPC to run training jobs and an Amazon S3 bucket to store trained model artifacts.

To help you resolve access issues, this section explains the AWS services and policies upon which AWS DeepRacer depends.

Topics

- [Sign Up Your AWS Account \(p. 6\)](#)
- [Create an IAM User \(p. 7\)](#)
- [AWS DeepRacer-Dependent AWS Services \(p. 7\)](#)
- [Required IAM Roles for AWS DeepRacer to Call Dependent AWS Services \(p. 9\)](#)

Sign Up Your AWS Account

To use AWS services, you need an AWS account. If you don't have one, sign up for one now.

The AWS account is free. You pay only for the AWS services that you use.

To sign up for an AWS account

1. Go to <https://console.aws.amazon.com>.
2. Choose **Create a Free Account**.
3. Follow the instructions on the page.

Part of the sign-up process involves receiving a phone call and entering a PIN by using the phone keypad.

If successful, the sign-up procedure creates an account with root user credentials, as identified by the email address that you provide. Although you can use this AWS account root user to sign in to AWS, we recommend that you create an IAM user with administrative privileges to manage your account. You can

also create IAM users with more restricted permissions. Use these for team members to call AWS services under your account. The instructions are given in the [next section \(p. 7\)](#).

Create an IAM User

The following procedure walks you through the steps to create an [AWS Identity and Access Management \(IAM\) user](#) with administrative privileges. To sign in to AWS DeepRacer and other services, we recommend that you don't use the root user of your AWS account.

To create an IAM user

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, then choose **Add user**.
3. For **Access type**, choose both **Programmatic Access** to call APIs of permitted AWS services and **AWS Management Console Access** to launch the AWS Management Console for permitted services.
4. For **Console password**, choose **Autogenerated password** or **Custom password**. If you choose **Custom password**, type a password.
5. Choose whether to require the user to reset the password at the next sign-in, then choose **Next: Permissions**.
6. For **Set permissions for <user name>**, choose **Attach existing policies directly**, **AdministrativeAccess**, and **Next: Review**.
7. Review the settings. To return to the previous page to make changes, choose **Previous**. To create the user, choose **Create user**.

For a non-administrative user to use AWS DeepRacer, follow the same steps to create an IAM user, but attach appropriate non-administrative access IAM roles and policies. If the role is not already created, the AWS DeepRacer service creates an appropriate IAM role with required policies upon request. You don't have to manually specify the access permission for AWS DeepRacer. For instructions to create the required roles and policies, see [Required IAM Roles for AWS DeepRacer to Call Dependent AWS Services \(p. 9\)](#).

Now that you have an AWS account and IAM user, you're ready to explore AWS DeepRacer. Learn reinforcement learning and explore its application to autonomous racing.

AWS DeepRacer-Dependent AWS Services

AWS DeepRacer uses the following AWS services to manage required resources:

Amazon Simple Storage Service

To store trained model artifacts in an Amazon S3 bucket.

AWS Lambda

To create and run the reward functions.

AWS CloudFormation

To create training jobs for AWS DeepRacer models.

Amazon SageMaker

To train the AWS DeepRacer models.

AWS RoboMaker

To simulate an environment for both training and evaluation.

The dependent AWS Lambda, AWS CloudFormation, Amazon SageMaker, and AWS RoboMaker in turn use other AWS services including Amazon CloudWatch and Amazon CloudWatch Logs.

The following table shows AWS services used by AWS DeepRacer, directly or indirectly.

AWS Services that AWS DeepRacer uses directly or indirectly

AWS service principal	Comments
application-autoscaling	<ul style="list-style-type: none"> Indirectly called by Amazon SageMaker to automatically scale its operations.
cloudformation	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to create training jobs for reinforcement learning models.
cloudwatch	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to log its operations. Indirectly called by AWS RoboMaker to log its operations. Indirectly called by Amazon SageMaker to log its operations.
ec2	<ul style="list-style-type: none"> Indirectly called by AWS CloudFormation and Amazon SageMaker to create and run training jobs.
ecr	<ul style="list-style-type: none"> Indirectly called by AWS RoboMaker to work with Amazon Elastic Container Registry.
kinesisvideo	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to view cached training streams. Indirectly called by AWS RoboMaker to cache training streams.
lambda	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to create and run the reward functions.
logs	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to log its operations. Indirectly called by AWS Lambda to log its operations. Indirectly called by AWS RoboMaker to log its operations.
robomaker	<ul style="list-style-type: none"> Directly called by AWS DeepRacer to render a virtual reinforcement learning environment in a simulation.
s3	<ul style="list-style-type: none"> Indirectly called by AWS RoboMaker to list a bucket beginning with 'deepracer' and to read objects in the bucket, or write objects to the bucket. Indirectly called by Amazon SageMaker to perform Amazon SageMaker-specific storage operations. Directly called by AWS DeepRacer to create, list, and delete buckets that have names starting with "deepracer." Also called to download

AWS service principal	Comments
	objects from the buckets, upload objects to the buckets, or delete objects from the buckets.
sagemaker	<ul style="list-style-type: none">Directly called by AWS DeepRacer to train reinforcement learning models.

To use AWS DeepRacer to call these services, you must have appropriate IAM roles with required policies attached to them. Learn the details about these policies and roles in [Required IAM Roles for AWS DeepRacer to Call Dependent AWS Services \(p. 9\)](#).

Required IAM Roles for AWS DeepRacer to Call Dependent AWS Services

Before you create a model, use the AWS DeepRacer console to set up resources for your account. As you do this, the AWS DeepRacer console creates the following IAM roles:

[AWSDeepRacerServiceRole](#)

Allows AWS DeepRacer to create required resources and call AWS services on your behalf.

[AWSDeepRacerSageMakerAccessRole](#)

Allows Amazon SageMaker to create required resources and call AWS services on your behalf.

[AWSDeepRacerRoboMakerAccessRole](#)

Allows AWS RoboMaker to create required resources and call AWS services on your behalf.

[AWSDeepRacerLambdaAccessRole](#)

Allows AWS Lambda functions to call AWS services on your behalf.

[AWSDeepRacerCloudFormationAccessRole](#)

Allows AWS CloudFormation to create and manage AWS stacks and resources on your behalf.

Follow the links to view detailed access permissions in the AWS IAM console.

Train Your First AWS DeepRacer Model for Autonomous Racing

Using the AWS DeepRacer console, you can follow built-in templates to train and evaluate an AWS DeepRacer model.

To train a reinforcement learning model for autonomous racing using the AWS DeepRacer console

1. Sign in to the [AWS DeepRacer console](https://console.aws.amazon.com/deepracer) (<https://console.aws.amazon.com/deepracer>).
2. On the AWS DeepRacer home page, choose **Get started** and do the following on the **Get started with reinforcement learning** page:
 - a. Under **Account resources**, choose **Create resources**, if this is your first time to use AWS DeepRacer.

- b. If you're new reinforcement learning, choose **Learn RL** under **Introduction to reinforcement learning** to get familiar with reinforcement learning.
- c. Under **Create a reinforcement learn (RL) model**, choose **Create model** to start to create your first AWS DeepRacer model.

If you have prior experiences with AWS DeepRacer, you can skip this step and choose instead **Reinforcement learning** on the primary navigation pane and then choose **Create model**.

3. On the **Create model** page, do the following to configure and start training your AWS DeepRacer model:
 - a. To refresh your account resources, choose **Reset resources** under **Account resources**. This action forces the required resources for your account to be recreated or re-associated.

When the required resources are created, you should see the following display:

Account resources [Info](#)

AWS DeepRacer requires permissions and resources in your AWS account to train and evaluate reinforcement learning models.

- You have valid IAM roles
- You have a valid S3 bucket
- You have a valid AWS DeepRacer VPC stack
- You have a valid RoboMaker simulation application

Having issues? Try resetting the resources created by AWS DeepRacer to start again from a clean slate. [Info](#)

Reset resources

The Amazon S3 bucket is used to store the trained model artifacts. And the IAM roles contain relevant IAM policies to grant AWS DeepRacer permission to call other AWS services on your behalf. For more information about the required IAM roles and policies, see [Required IAM Roles for AWS DeepRacer to Call Dependent AWS Services \(p. 9\)](#). The AWS DeepRacer Virtual Private Cloud (VPC) stack is used to run training jobs. The AWS RoboMaker simulation application is used to visualize training and evaluating your model.

- b. Under **Model details**, enter a name for the to-be-trained model in the **Model name** input field and optionally provide a brief description in the **Model description - optional** input field.

Model details

Model name

my-deepracer-model

The model name must be unique and can have up to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces or underscores.

Model description - *optional*

My AWS deepRacer model

The model description can have up to 255 characters.

The model name identifies this model among the AWS DeepRacer models you've created and on the leaderboards, which show the performance rank of the model. The model description can provide a summary of the model features and limitations.

- c. Under **Environment simulation**, choose an available track as a virtual environment to train your AWS DeepRacer model through trials and errors. For your first run, choose a simple option, e.g., of **Straight track**. In later iterations, you can choose more complex track to progressively improve your models. To train a model for a particular racing event, choose a track most similar to the track of the event.
- d. Under **Action space**, use the default setting for your first training. Later, you can explore different size and granularity of the action space to make your model sufficiently flexible to accommodate the selected track.
- e. Under **Reward function**, use the default reward function example or choose **Reward function examples** to select another example function and then choose **Use code** of the selected function.

There are three example functions you can start with. They illustrate how to follow the track center (default), how to keep the agent inside the track borders, and how to prevent zig-zag driving.

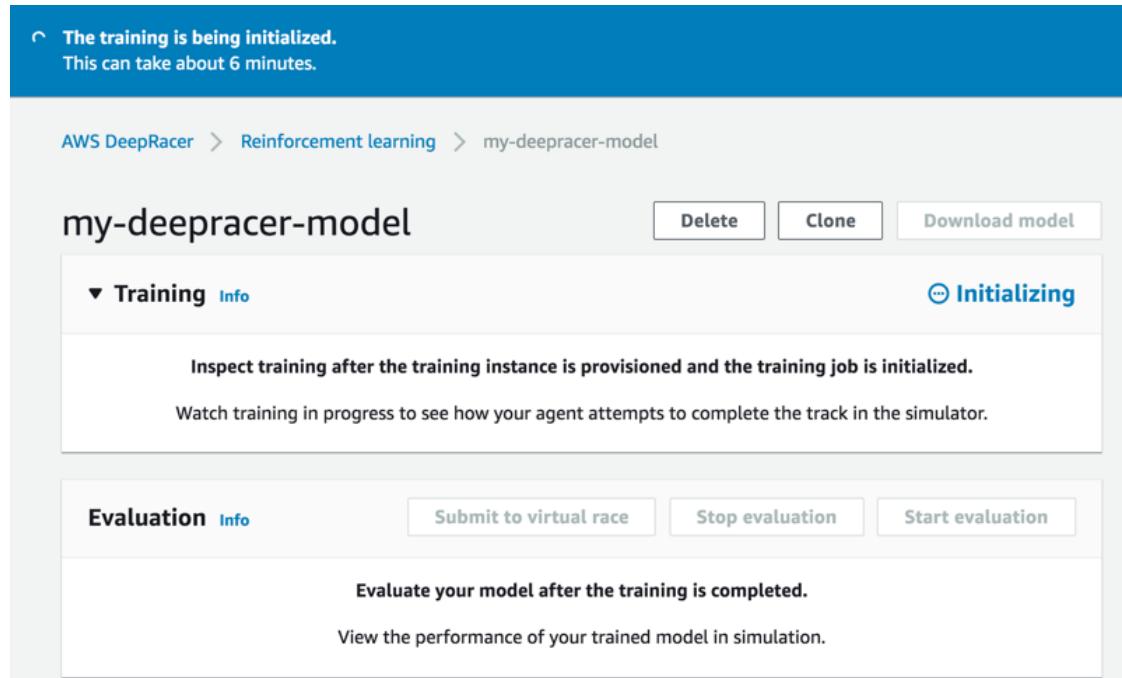
After getting familiar with the reward function examples, you can modify or replace the predefined reward function code in the code editor.

- f. Under **Hyperparameters**, you can leave the default values or expand the section to set **Hyperparameters** values as follows:
 - i. For **Gradient descent batch size**, choose [available options \(p. 34\)](#).
 - ii. For **Number of epochs**, set a [valid value \(p. 34\)](#).
 - iii. For **Learning rate**, set a [valid value \(p. 34\)](#).
 - iv. For **Entropy**, set a [valid value \(p. 34\)](#).
 - v. For **Discount factor**, set a [valid value \(p. 34\)](#).
 - vi. For **Loss type**, choose [available options \(p. 34\)](#).
 - vii. For **Number of experience episodes between each policy-updating iteration**, set a [valid value \(p. 34\)](#).

For more information about hyperparameters, see [Systematically Tune Hyperparameters \(p. 34\)](#).

- g. Under **Stop conditions**, set a **Maximum time** value to terminate long-running (and possible run-away) training session.

- h. Choose **Start training** to start creating the model and provisioning the training job instance.
4. A training starts with provisioning a AWS DeepRacer training job. The process takes about 6 minutes and its status is **Initializing** until the training job is successfully provisioned.



5. After the job is provisioned, training starts and the process has **In progress** status until the job stops. You can choose the refresh button to periodically refresh the **Reward graph**.

Training in progress
Training will stop after 60 minutes.

Stop training

AWS DeepRacer > Reinforcement learning > my-deepracer-model

my-deepracer-model

Delete Clone Download model

▼ Training [Info](#) ⌚ In progress

AWS DeepRacer leverages Amazon SageMaker to train your model behind the scenes and uses AWS RoboMaker to simulate the agent's interaction with the environment. Watch how the agent behaves in the chosen environment, as prescribed by the reward function.

Reward graph ↻ 2019-04-22 (14:45:20) - 2019-04-22 (15:45:20) ▾
Total reward over time

Simulation video stream

Stop condition
Maximum time
00:55:50 / 01:00:00

- After the training job stops, the process is **Completed**, and you can start to [evaluate your trained model \(p. 14\)](#) to measure its performance.

Training completed. You can now evaluate your model.

AWS DeepRacer > Reinforcement learning > my-deepracer-model-2

my-deepracer-model-2

Delete Clone Download model

▼ Training Info Completed

AWS DeepRacer leverages Amazon SageMaker to train your model behind the scenes and uses AWS RoboMaker to simulate the agent's interaction with the environment. Watch how the agent behaves in the chosen environment, as prescribed by the reward function.

Reward graph Simulation video stream

2019-04-22 (14:45:20) - 2019-04-22 (15:45:20) ↻

Total reward over time

Stop condition

Maximum time

01:00:00 / 01:00:00

Simulation video stream

Simulation video stream not available.
Video is only available during training.

Evaluate Your AWS DeepRacer Models in Simulation

After your model is trained, use the AWS DeepRacer console to evaluate its performance. In AWS DeepRacer, the performance metric is the time required to complete a track, following the inferred actions as prescribed by the trained model.

To evaluate a trained model using the AWS DeepRacer console, follow the steps below.

1. On your model's details page, under the **Evaluation** section, choose **Start evaluation**.

You can start an evaluation after your model is in a **Ready** state. A model is ready when the training is complete. If the training is not complete, the model might also be in a ready state if its trained up to the failing point.

The screenshot shows the AWS DeepRacer Developer Guide page with the title "Evaluate Models in Simulation". Below the title, there is a green banner with the text "Training completed. You can now evaluate your model.". Underneath the banner, the navigation path is shown as "AWS DeepRacer > Reinforcement learning > my-deepracer-model". The main content area displays a model named "my-deepracer-model". On the right side of the model name, there are three buttons: "Delete", "Clone", and "Download model". Below the model name, there are two tabs: "Training" (with a "Completed" status) and "Evaluation". The "Evaluation" tab is currently selected. Under the "Evaluation" tab, there are three buttons: "Submit to virtual race", "Stop evaluation", and a prominent orange "Start evaluation" button, which is circled in red. Below these buttons, there is a message: "Training completed. You can now evaluate your model." and a sub-instruction: "View the performance of your trained model in simulation."

2. Under **Select environment to start evaluation**, choose an evaluation track.

Select environments to start evaluation X

The trained model will be evaluated in specified simulated environment.

Choose an option

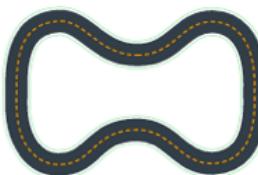
re:Invent 2018

Inspired by the classic track at Monza, the re:Invent track was the first track on which we tested and launched AWS DeepRacer.



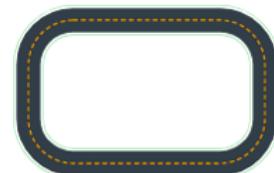
Bowtie Track

Inspired by a quest to find a simple symmetrical track with a twist, the Bowtie delivers just that. The track features shallow turns that will put your car's racing lines to the test.



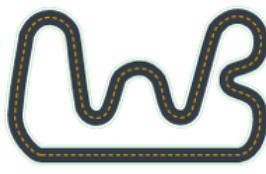
Oval Track

We went looking for a track that was simple and gave the car some space to play, and found inspiration from the Indy Speedway.



AWS Track

Seeking a track that really put our cars cornering at speed to the test, our designers dreamt the AWS track. This track is far larger than others and training time may take longer.



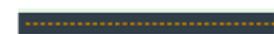
H Track

Another larger than normal track, the H track was built to see whether your car could learn to accelerate and slow down.



Straight Track

The Straight offers a simple strip on which you can quickly train models to race in a straight line. Most tracks, however, have corners... so use this one at your own risk.



Specify stop condition

Your reinforcement model will evaluate until specified criteria has been met or by manually stopping the evaluation.

Number of trials

3 trials

Cancel

Start evaluation

Typically, you want to choose a track that is the same as or similar to the one you used in [training the model \(p. 9\)](#). You can choose any track for evaluating your model, however, you can expect the best performance on a track most similar to the one used in training.

3. To specify the stop condition, leave the default value (3 trials) as-is for **Number of trials**. You can specify 3-5 trial runs for each evaluation.
4. Choose **Start evaluation** to start creating and initializing the evaluation job.

This initialization process takes about 3 minutes to complete.

The screenshot shows the AWS DeepRacer console interface. At the top, a blue banner displays the message: "Evaluation job is being initialized. The evaluation job is being initialized and it will take about 3 minutes." Below the banner, the navigation path is shown as: AWS DeepRacer > Reinforcement learning > my-deepracer-model. The main title "my-deepracer-model" is displayed, along with three buttons: Delete, Clone, and Download model. Under the "Training" tab, which is currently selected (indicated by a blue border), the status is "Completed". In the "Evaluation" tab, which is also selected, there are three buttons: Submit to virtual race, Stop evaluation, and Start new evaluation. The "Evaluation results" section is currently empty, showing the message "No evaluation trials yet" and "Evaluation job status: Initializing". The "Simulation video stream" section also indicates that no video is available during evaluation. The overall status of the evaluation job is "Initializing".

5. After evaluation is in progress, choose **Stop evaluation** if you would like to stop the evaluation for any reason.

Evaluation in progress.
You will see per-trial performance as the evaluation progresses.

AWS DeepRacer > Reinforcement learning > my-deepracer-model

my-deepracer-model

Delete Clone Download model

▶ Training [Info](#) 🕒 Completed

Evaluation [Info](#) Submit to virtual race Stop evaluation Start new evaluation

Simulation video stream Evaluation results



Trial	Time	Trial results (% track completed)
1	00:01:30.868	100%

To stop an evaluation job, choose **Stop evaluation** on the upper-right corner of the **Evaluation** pane and then confirm to stop the evaluation.

6. After evaluation completed successfully, inspect the **Evaluation results** to see how your model performs.

The screenshot shows the AWS DeepRacer console interface. At the top, the navigation path is "AWS DeepRacer > Reinforcement learning > my-deepracer-model". The main title is "my-deepracer-model". On the right, there are buttons for "Delete", "Clone", and "Download model". Below the title, a section for "Training" is shown with a "Completed" status indicator. A large "Evaluation" section follows, containing buttons for "Submit to virtual race", "Stop evaluation", and "Start new evaluation". The "Evaluation results" table has columns for "Trial", "Time", and "Trial results (% track completed)". Three trials are listed: Trial 1 (Time: 00:01:30.868, Result: 100%), Trial 2 (Time: 00:01:28.937, Result: 100%), and Trial 3 (Time: 00:01:30.849, Result: 100%). To the left of the results table, a placeholder for a "Simulation video stream" is shown with a message: "Simulation video stream not available. Video is only available during evaluation."

Trial	Time	Trial results (% track completed)
1	00:01:30.868	100%
2	00:01:28.937	100%
3	00:01:30.849	100%

The **Time** values listed under **Evaluation results** show how fast a trial lasts from the start position to the finish line or getting off-track. The **Trial results (% completed)** value shows the percentage of the track that was completed. A 100% value means the trial completed successfully.

Anything less than 100 percent means the trial failed. A model with a less than 100 percent trial result is not ready for racing. You can anticipate such failure when the [total reward \(p. 29\)](#) in the training doesn't appear to have converged. In this case, you can continue to improve the model by cloning it, changing the reward function, tuning hyperparameters, and then iterating the process until the total reward converges and the performance metrics improve. For more information on how to improve the training, see [Train and Evaluate Models \(p. 20\)](#).

To transfer your completely trained model to your AWS DeepRacer vehicle for driving in a physical environment, you need to first download the model artifacts. To do so, choose **Download model** on the model's details page. For more information about testing an AWS DeepRacer model with a physical agent, see [Operate Your Vehicle \(p. 70\)](#).

If you've trained your model on a track identical or similar to the one chosen in a racing event of the AWS DeepRacer League, you can submit the model to the event. Do this by choosing **DeepRacer League** from the primary navigation pane on the AWS DeepRacer console. For more information, see [Rank Models in Leaderboard \(p. 109\)](#).

Train and Evaluate AWS DeepRacer Models

When your AWS DeepRacer vehicle drives itself along a track, it captures environmental states with the camera mounted on the front and takes actions in response to the observations. Your AWS DeepRacer model is a function that maps the observations and actions to the expected reward. To train your model is to find or learn the function that maximize the expected reward so that the optimized model prescribes what actions (speed and steering angle pairs) your vehicle can take to move itself along the track from start to finish.

In practice, the function is represented by a neural network and training the network involves finding the optimal network weights given sequences of observed environmental states and the responding vehicle's actions. The underlying criteria of optimality are described by the model's reward function that encourages the vehicle to make legal and productive moves without causing traffic accidents or infractions. A simple reward function could return a reward of 0 if the vehicle is on the track, -1 if it's off the track, and +1 if it reaches the finish line. With this reward function, the vehicle gets penalized for going off the track and rewarded for reaching the destination.

For example, let's say that you're interested in having the vehicle drive without getting off a straight track. As the vehicle speeds up and down, the vehicle may steer left or right to avoid obstacles or to remain inside. For timed runs, you want the vehicle to go faster. But making too big a turn at a high speed could easily lead the vehicle off the track. Making too small a turn may not help avoid colliding with an obstacle or another vehicle. Generally speaking, optimal actions would be to make a bigger turn at a lower speed or to steer less along a sharper curve. To encourage this behavior, your reward function must assign a positive score to reward smaller turns at a higher speed and/or a negative score to punish bigger turns at a higher speed. Similarly, the reward function can return a positive reward for speeding up along a straighter course or speeding down when it's near an obstacle.

The reward function is an important part of your AWS DeepRacer model. You must provide it when training your AWS DeepRacer model. The training involves repeated episodes along the track from start to end. In an episode the agent interacts with the track to learn the optimal course of actions by maximizing the expected cumulative reward. At the end, the training produces a reinforcement learning model. After the training, the agent executes autonomous driving by running inference on the model to take an optimal action in any given state. This can be done in either the simulated environment with a virtual agent or a real-world environment with a physical agent, such as an AWS DeepRacer scale vehicle.

To train a reinforcement learning model in practice, you must choose a learning algorithm. Currently, the AWS DeepRacer console supports only the proximal policy optimization ([PPO](#)) algorithm for faster training performances. You can then choose a deep-learning framework supporting the chosen algorithm, unless you want to write one from scratch. AWS DeepRacer integrates with Amazon SageMaker to make some popular deep-learning frameworks, such as [TensorFlow](#), readily available in the AWS DeepRacer console. Using a framework simplifies configuring and executing training jobs and lets you focus on creating and enhancing reward functions specific to your problems.

Training a reinforcement learning model is an iterative process. First, it's a challenge to define a reward function to cover all important behaviors of an agent in an environment at once. Second, hyperparameters are often tuned to ensure satisfactory training performance. Both require experimentation. A prudent approach is to start with a simple reward function and then progressively enhance it. AWS DeepRacer facilitates this iterative process by enabling you to clone a trained model and then use it to jump-start the next round training. At each iteration you can introduce one or a few

more sophisticated treatments to the reward function to handle previous ignored variables or you can systematically adjust hyperparameters until the result converges.

As with general practice in machine learning, you must evaluate a trained reinforcement learning model to ascertain its efficacy before deploying it to a physical agent for running inference in a real-world situation. For autonomous driving, the evaluation can be based on how often a vehicle stays on a given track from start to finish or how fast it can finish the course without getting off the track. The AWS DeepRacer simulation runs in the AWS RoboMaker simulator and lets you run the evaluation and post the performance metrics for comparison with models trained by other AWS DeepRacer users on a [leaderboard \(p. 109\)](#).

Topics

- [Understand Reinforcement Learning in AWS DeepRacer \(p. 21\)](#)
- [Train and Evaluate AWS DeepRacer Models Using the AWS DeepRacer Console \(p. 28\)](#)
- [Train and Evaluate AWS DeepRacer Models Using Amazon SageMaker Notebooks \(p. 42\)](#)
- [Input Parameters of the AWS DeepRacer Reward Function \(p. 57\)](#)

Understand Reinforcement Learning in AWS DeepRacer

AWS DeepRacer uses reinforcement learning to enable the AWS DeepRacer 1/18th scale vehicle to drive autonomously. To achieve this, you train and evaluate a reinforcement learning model in a virtual environment with a simulated track. After the training, you upload the trained model artifacts to your AWS DeepRacer vehicle. You can then set the vehicle for autonomous driving in a physical environment with a real track.

Training a reinforcement learning model can be challenging, especially if you're new to the field. AWS DeepRacer simplifies the process by integrating required components together and providing easy-to-follow wizard-like task templates. However, it's helpful to have a good understanding of the basics of reinforcement learning training implemented in AWS DeepRacer.

Topics

- [Overview of Reinforcement Learning \(p. 21\)](#)
- [AWS DeepRacer Action Space and Reward Function \(p. 23\)](#)
- [AWS DeepRacer Training Algorithm \(p. 24\)](#)
- [AWS DeepRacer Service Architecture \(p. 26\)](#)
- [AWS DeepRacer Solution Workflow \(p. 27\)](#)
- [Simulated-to-Real Performance Gaps \(p. 28\)](#)

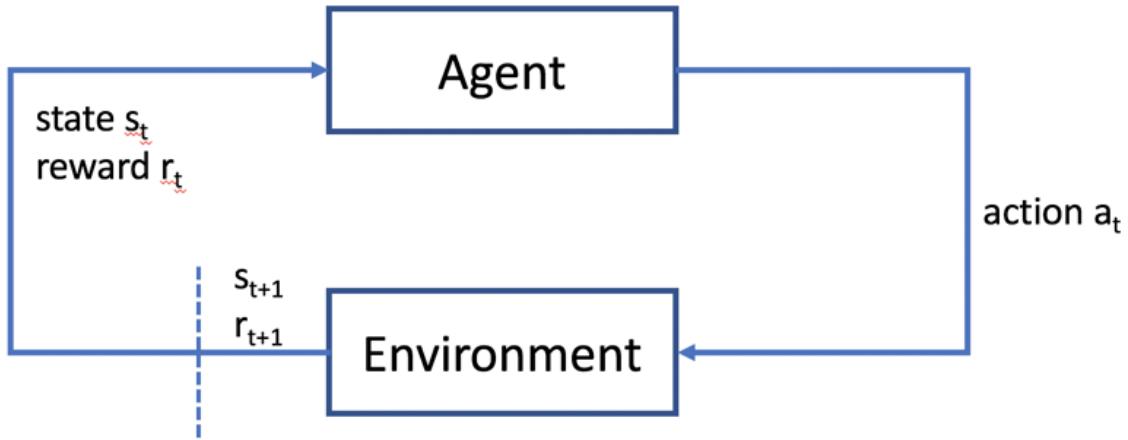
Overview of Reinforcement Learning

In reinforcement learning, an *agent* with an objective interacts with an *environment* to maximize the agent's total reward. The agent takes an *action*, guided by a strategy referred to as a *policy*, at a given environment *state* and reaches a new state. There is an immediate *reward* associated with any action. The reward is a measure of the desirability of the action. This immediate reward is considered to be returned by the environment.

The goal of the reinforcement learning in AWS DeepRacer is to learn the optimal policy in a given environment. Learning is an iterative process of trials and errors. The agent takes the random initial

action to arrive at a new state. Then the agent iterates the step from the new state to the next one. Over time, the agent discovers actions that lead to the maximum long-term rewards. The interaction of the agent from an initial state to a terminal state is called an *episode*.

The following sketch illustrates this learning process:



The *agent* embodies a neural network that represents a function to approximate the agent's policy. The image from the vehicle's front camera is the environment *state* and the agent *action* is defined by the agent's speed and steering angles.

The agent receives positive *rewards* if it stays on-track to finish the race and negative rewards for going off-track. An *episode* starts with the agent somewhere on the race track and finishes when the agent either goes off-track or completes a lap.

Note

Strictly speaking, the environment state refers to everything relevant to the problem. For example, the vehicle's position on the track as well as the shape of the track. The image fed through the camera mounted the vehicle's front does not capture the entire environment state. Hence, the environment is deemed partially observed and the input to the agent is referred to as *observation* rather than state. For simplicity, we use *state* and *observation* interchangeably throughout this documentation.

Training the agent in a simulated environment has the following advantages:

- The simulation can estimate how much progress the agent has made and identify when it goes off the track to compute a reward.
- The simulation relieves the trainer from tedious chores to reset the vehicle each time it goes off the track, as is done in a physical environment.
- The simulation can speed up training.
- The simulation provides better controls of the environment conditions, e.g. selecting different tracks, backgrounds, and vehicle conditions.

The alternative to reinforcement learning is *supervised learning*, also referred to as *imitation learning*. Here a known dataset (of [image, action] tuples) collected from a given environment is used to train the agent. Models that are trained through imitation learning can be applied to autonomous driving. They work well only when the images from the camera look similar to the images in the training dataset. For robust driving, the training dataset must be comprehensive. In contrast, reinforcement learning does not require such extensive labeling efforts and can be trained entirely in simulation. Because reinforcement learning starts with random actions, the agent learns a variety of environment and track conditions. This makes the trained model robust.

AWS DeepRacer Action Space and Reward Function

For autonomous driving, the AWS DeepRacer vehicle receives input images streamed at 15 frames per second from the front camera. The raw input is downsized to 160x120 pixels in size and converted to grayscale images.

Responding to an input observation, the vehicle reacts with a well-defined action of specific speed and steering angle. The actions are converted to low-level motor controls. The possible actions a vehicle can take is defined by an action space of the dimensions in speed and steering angle. An action space can be discrete or continuous. AWS DeepRacer uses a discrete action space.

For a discrete action space of finite actions, the range is defined by the maximum speed and the absolute value of the maximum steering angles. The granularities define the number of speeds and steering angles the agent has.

For example, the AWS DeepRacer default action space has the following six actions you can use to train an AWS DeepRacer model.

The default AWS DeepRacer action space

Action number	Steering	Speed
0	-30 degrees	0.4 m/s
1	-30 degrees	0.8 m/s
2	-15 degrees	0.4 m/s
3	-15 degrees	0.8 m/s
4	0 degrees	0.4 m/s
5	0 degrees	0.8 m/s
6	15 degrees	0.4 m/s
7	15 degrees	0.8 m/s
8	30 degrees	0.4 m/s
9	30 degrees	0.8 m/s

This default action space is characterized by the following ranges and granularities:

The default action space characteristics

Property	Value	Description
Maximum steering angle	30 degrees	
Steering angle granularity	5	
Maximum speed	0.8 m/s	
Speed granularity	2	

To influence behavior, we can explore a reward function to assign immediate rewards to the actions in this action space. For example, AWS DeepRacer has a basic reward function by default to encourage the

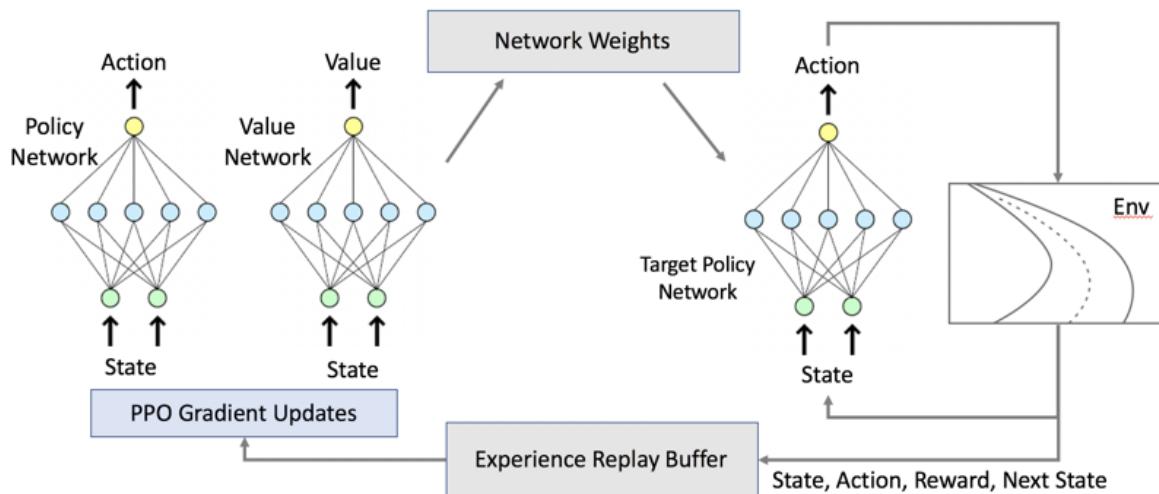
agent to stay as close to the center line as possible. The agent avoids steering close to the edge of the track and going off the track with even a slight turn. For details of this default reward function in the AWS DeepRacer console, see [AWS DeepRacer reward function example \(p. 31\)](#).

In addition to the default action space, you can also explore a [custom action space \(p. 33\)](#) and a [custom reward function \(p. 29\)](#) to train your models.

You can apply a trained model to an AWS DeepRacer vehicle by mapping the maximum speed (0.8 m/s) and maximum steering angles (30 degrees) used in training to the corresponding maximum physical values. This is called [vehicle calibration \(p. 85\)](#).

AWS DeepRacer Training Algorithm

AWS DeepRacer uses the [Proximal Policy Optimization \(PPO\)](#) algorithm to train the reinforcement learning model. PPO uses two neural networks during training: a policy network and a value network. The policy network (also called actor network) decides which action to take given an image as input. The value network (also called critic network) estimates the cumulative reward we are likely to get given the image as input. Only the policy network interacts with the simulator and gets deployed to the real agent, namely an AWS DeepRacer vehicle.



Below we explain how the actor and critic work together mathematically.

PPO is a derivative of the policy gradient method. In the most basic form, the policy gradient method trains the agent to move along a track by searching for the optimal policy $\#^*(a|s; \#^*)$. The optimization aims at maximizing a policy score function $J(\#)$ that can be expressed in terms of the immediate reward $r(s, a)$ of taking action (a) in state (s) averaged over the state probability distribution $\#(s)$ and the action probability distribution $(\#(a|s; \#))$:

$$J(\theta) = \sum_{s \in S} \rho(s) \sum_{a \in A} \pi(a|s; \theta) r(s, a)$$

The optimal policy, as represented by the optimal policy network weights $\#^*$, can be expressed as follows:

$$\theta^* = \operatorname{argmax}_{\theta} (J(\theta))$$

The maximization can proceed by following the policy gradient ascent over episodes of training data (s , a , r):

$$\theta_{\tau+1} = \theta_{\tau} + \alpha \nabla_{\theta} J(\theta)$$

where α is known as the learning rate and $\nabla_{\theta} J(\theta)$ is the policy gradient with respect to θ evaluated at step τ .

In terms of the total future reward:

$$R(\tau) = R(s_{\tau}, r_{\tau}) = \sum_{t=0}^H \gamma^t r(s_{\tau+t}, r_{\tau+t})$$

where γ is the discount factor ranging between 0 and 1, and τ maps to an *experience* $(s_{\tau}, a_{\tau}, r_{\tau})$ at step τ , and the summation includes experiences in an episode that starts from time $t = 0$ and ends at time $t = H$ when the agent goes off-track or reaches to the finish line, the score function becomes the expected total future reward averaged over the policy distribution π across many episodes of experiences:

$$J(\theta) = \sum_{\tau} \pi(\theta, \tau) R(\tau)$$

From this definition of $J(\theta)$, the policy weight updates can be expressed as follows:

$$\nabla_{\theta} J(\theta_{\tau}) = \sum_{\tau} \pi(\theta, \tau) \nabla_{\theta} \log(\pi(\theta, \tau)) R(\tau) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^H \nabla_{\theta} \log(\pi(a_{i,t}|s_{i,t}; \theta)) R(a_{i,t}, s_{i,t})$$

Here, averaging over π is approximated by sample averaging over N of episodes each of which consists of possibly unequal H number of experiences.

The update rule for a policy network weight then becomes:

$$\Delta \theta_{i,t} = \alpha \nabla_{\theta} \log(\pi(a_{i,t}|s_{i,t}; \theta)) R(a_{i,t}, s_{i,t})$$

The policy gradient method outlined above is of limited utility in practice, because the score function $R(s_{i,t}, a_{i,t})$ has high variance as the agent can take many different paths from a given state. To get around this, one uses a critic network (#) to estimate the score function. To illustrate this, let $V_\#(s)$ the value of the critic network, where s describes a state and $\#$ the value network weights. To train this value network, the estimated value ($y_{i,t}$) of state s at step t in episode i is estimated to be the immediate reward taking action $a_{i,t}$ at state $s_{i,t}$ plus the discounted total future value of the state s at the next step $t+1$ in the same episode:

$$y_{i,t} \approx r(a_{i,t}, s_{i,t}) + \gamma V_\phi(s_{i,t+1})$$

The loss function for the value network weights is:

$$L(\phi) = \frac{1}{2} \sum_{i,t} (V_\phi(s_{i,t}) - y_{i,t})^2$$

Using the estimated values, the policy gradient for updating the policy network weights # becomes:

$$\nabla_\theta J(\theta_\tau) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^H \nabla_\theta \log(\pi(a_{i,t}|s_{i,t}; \theta)) (r(a_{i,t}, s_{i,t}) + \gamma V_\phi(s_{i,t+1}) - V_\phi(s_{i,t}))$$

This formulation changes the policy network weights such that it encourages actions that give higher rewards than prior estimate and discourages otherwise.

Every reinforcement learning algorithm needs to balance between exploration and exploitation. The agent needs to explore the state and action space to learn which actions lead to high rewards in unexplored state space. The agent should also exploit by taking the actions that leads to high rewards so that the model converges to a stable solution. In our algorithm, the policy network outputs the probability of taking each action and during training the action is chosen by sampling from this probability distribution (e.g. an action with probability 0.5 will be chosen half the time). During evaluation, the agent picks with action with the highest probability.

In addition to the above actor-critic framework, PPO uses importance sampling with clipping, adds a [Gauss-Markov noise](#) to encourage exploration and uses [generalized advantage estimation](#). To learn more, see the [original paper](#).

AWS DeepRacer Service Architecture

The AWS DeepRacer service is built upon [Amazon SageMaker](#), [AWS RoboMaker](#) and other AWS services such as [Amazon S3](#).

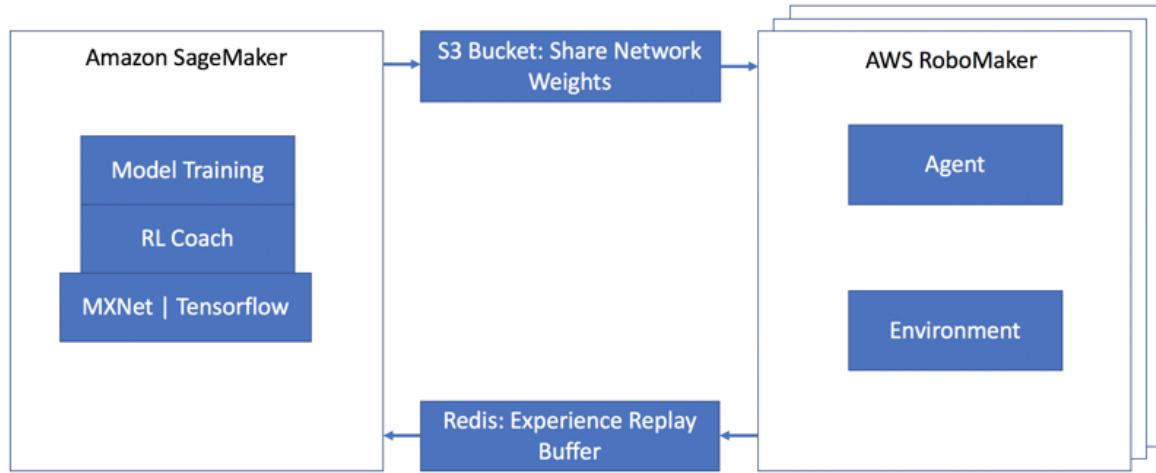
Amazon SageMaker is an AWS machine learning platform to train machine learning models in general. AWS DeepRacer uses it to train reinforcement learning model in particular. AWS RoboMaker is a cloud service to develop, test and deploy robotic solutions in general. AWS DeepRacer uses it to create the virtual agent and its interactive environment. Amazon S3 is an economical general-purpose cloud storage solution. AWS DeepRacer uses it to store trained model artifacts. In addition, AWS DeepRacer uses [Redis](#), an in-memory database, as an experience buffer to select training data from for training the policy neural network.

Within the AWS DeepRacer architecture, AWS RoboMaker creates a simulated environment for the agent to drive along a specified track. The agent moves according to the policy network model that has been

trained up to a certain time in Amazon SageMaker. Each run starts from the starting line to an end state which can be the finishing line or off the track and is known as an episode. For each episode, the course is divided into segments of a fixed number of steps. In each segment experiences, defined as an ordered list of the tuples of (*state, action, reward, new state*) associated with individual steps, are cached in Redis as an experience buffer. Amazon SageMaker then randomly draws from the experience buffer training data in batches and feeds the input data to the neural network to update the weights. It then stores the updated model in Amazon S3 for Amazon SageMaker to use in order to generate more experiences. The cycle continues until training stops.

In the beginning before the first model is trained for the first time, Amazon SageMaker initializes the experience buffer with random actions.

The following diagram illustrates this architecture.



This setup allows running multiple simulations to train a model on multiple segments of a single track at the same time or to train the model for multiple tracks simultaneously.

AWS DeepRacer Solution Workflow

Training an AWS DeepRacer model involves the following general tasks:

1. The AWS DeepRacer service initializes the simulation with a virtual track, an agent representing the vehicle, and the background. The agent embodies a policy neural network that can be tuned with hyper-parameters as defined in the [PPO algorithm \(p. 24\)](#).
2. The agent acts (as specified with a steering angle and a speed) based on a given state (represented by an image from the front camera).
3. The simulated environment updates the agent's position based on the agent action and returns a reward and an updated camera image. The experiences collected in the form of state, action, reward, and new state are used to update the neural network periodically. The updated network models are used to create more experiences.
4. You can monitor the training in progress along the simulated track with a first-person view as seen by the agent. You can display metrics such as rewards per episode, the loss function value, the entropy of the policy. CPU or memory utilization can also be displayed as training progresses. In addition, detailed logs are recorded for analysis and debugging.
5. The AWS DeepRacer service periodically saves the neural network model to persistent storage.
6. The training stops based on a time limit.
7. You can evaluate the trained model in a simulator. To do this submit the trained model for time trials for a selected number runs on the selected track.

After the model is successfully trained and evaluated, it can be uploaded to a physical agent (an AWS DeepRacer vehicle). The process involves the following steps:

1. Download the trained model from its persistent storage (an Amazon S3 bucket).
2. Use the vehicle's device control console to upload the trained model to the vehicle. Use the console to calibrate the vehicle for mapping the simulated action space to the physical action space. You can also use the console to check the throttling parity, view the front camera feed, load a model into the inference engine, and watch the vehicle driving on a real track.

The vehicle's device control console is a web server hosted on the vehicle's compute module. The console is accessible from the vehicle IP address with a connected Wi-Fi network and a web browser on a computer or a mobile device.

3. Experiment with the vehicle driving under different lighting, battery levels, and surface textures and colors.

The vehicle's performance in a physical environment may not match the performance in a simulated environment due to model limitations or insufficient training. The phenomenon is referred to as the *sim2real* performance gap. To reduce the gap, see [the section called "Simulated-to-Real Performance Gaps" \(p. 28\)](#).

Simulated-to-Real Performance Gaps

Because the simulation cannot capture all aspects of the real world accurately, the models trained in simulation may not work well in the real world. Such discrepancies are often referred to as simulated-to-real (*sim2real*) performance gaps.

Efforts have been made in AWS DeepRacer to minimize the *sim2real* performance gap. For example, the simulated agent is programmed to take about 10 actions per second. This matches the frequency the AWS DeepRacer vehicle runs inference with, about 10 inferences per second. As another example, at the start of each episode in training, the agent's position is randomized. This maximizes the likelihood that the agent learns all parts of the track evenly.

To help reduce *real2sim* performance gaps, make sure to use the same or similar color, shape and dimensions for both the simulated and real tracks. To reduce visual distractions, use barricades around the real track. Also, carefully calibrate the ranges of the vehicle's speed and steering angles so that the action space used in training matches the real world. Evaluating model performance in a different simulation track than the one used in training can show the extent of the *real2real* performance gap.

For more information about how to reduce the *sim2real* gap when training an AWS DeepRacer model, see [the section called "Optimize Training for Real Environments" \(p. 40\)](#).

Train and Evaluate AWS DeepRacer Models Using the AWS DeepRacer Console

To train a reinforcement learning model, you can use the AWS DeepRacer console. In the console, create a training job, choose a supported framework and an available algorithm, add a reward function, and configure training settings. You can also watch training proceed in a simulator. You can find the step-by-step instructions in [the section called "Train Your First Model" \(p. 9\)](#).

This section explains how to train and evaluate an AWS DeepRacer model. It also shows how to create and improve a reward function, how an action space affects model performance, and how hyperparameters affect training performance. You can also learn how to clone a training model to extend a training session, how to use the simulator to evaluate training performance, and how to address some of the simulation to real-world challenges.

Topics

- [Create Your Reward Function \(p. 29\)](#)
- [Explore Action Space to Train a Robust Model \(p. 33\)](#)
- [Systematically Tune Hyperparameters \(p. 34\)](#)
- [Clone a Trained Model to Start a New Training Pass \(p. 37\)](#)
- [Evaluate AWS DeepRacer Models in Simulations \(p. 38\)](#)
- [Log AWS DeepRacer Events to CloudWatch Logs \(p. 38\)](#)
- [Optimize Training AWS DeepRacer Models for Real Environments \(p. 40\)](#)

Create Your Reward Function

A reward function describes immediate feedback (as a reward or penalty score) when your AWS DeepRacer vehicle moves from one position on the track to a new position. The function's purpose is to encourage the vehicle to make moves along the track to reach a destination quickly without accident or infraction. A desirable move earns a higher score for the action or its target state. An illegal or wasteful move earns a lower score. When training an AWS DeepRacer model, the reward function is the only application-specific part.

In general, you design your reward function to act like an incentive plan. Different incentive strategies could result in different vehicle behaviors. To make the vehicle drive faster, the function should give rewards for the vehicle to follow the track. The function should dispense penalties when the vehicle takes too long to finish a lap or goes off the track. To avoid zig-zag driving patterns, it could reward the vehicle to steer less on straighter portions of the track. The reward function might give positive scores when the vehicle passes certain milestones, as measured by [waypoints \(p. 57\)](#). This could alleviate waiting or driving in the wrong direction. It is also likely that you would change the reward function to account for the track conditions. However, the more your reward function takes into account environment-specific information, the more likely your trained model is over-fitted and less general. To make your model more generally applicable, you can explore [action space \(p. 33\)](#).

If an incentive plan is not carefully considered, it can lead to [unintended consequences of opposite effect](#). This is possible because the immediate feedback is a necessary but not sufficient condition for reinforcement learning. An individual immediate reward by itself also can't determine if the move is desirable. At a given position, a move can earn a high reward. A subsequent move could go off the track and earn a low score. In such case, the vehicle should avoid the move of the high score at that position. Only when all future moves from a given position yield a high score on average should the move to the next position be deemed desirable. Future feedback is discounted at a rate that allows for only a small number of future moves or positions to be included in the average reward calculation.

A good practice to create a reward function is to start with a simple one that covers basic scenarios. You can enhance the function to handle more actions. Before you learn about various reward function examples, let's let a look at the reward function signature.

Topics

- [The AWS DeepRacer Reward Function Signature \(p. 29\)](#)
- [Simple Reward Function Examples \(p. 30\)](#)
- [Enhance Your Reward Function \(p. 31\)](#)

The AWS DeepRacer Reward Function Signature

The AWS DeepRacer reward function has the following signature:

```
def reward_function(params) :  
    reward = ...
```

```
    return float(reward)
```

where `params` is a dictionary of the following form:

```
{
    "all_wheels_on_track (p. 58)": Boolean,           # flag to indicate if the vehicle is on the
    track
    "x (p. 68)": float,                             # vehicle's x-coordinate in meters
    "y (p. 68)": float,                             # vehicle's y-coordinate in meters
    "distance_from_center (p. 61)": float,           # distance in meters from the track center
    "is_left_of_center (p. 63)": Boolean,             # Flag to indicate if the vehicle is on the
    left side to the track center or not.
    "heading (p. 63)": float,                         # vehicle's yaw in degrees
    "progress (p. 63)": float,                        # percentage of track completed
    "steps (p. 66)": int,                            # number steps completed
    "speed (p. 64)": float,                          # vehicle's speed in meters per second (m/
    s)
    "steering_angle (p. 64)": float,                 # vehicle's steering angle in degrees
    "track_width (p. 66)": float,                     # width of the track
    "waypoints (p. 68)": [[float, float], ...],       # list of [x,y] as milestones along the
    track center
    "closest_waypoints (p. 60)": [int, int]          # indices of the two nearest waypoints.
}
```

The `reward_function` outputs the immediate reward (`reward`) at the specified step and the output is a number in the range of [-100000.0 : 100000.0].

Note

When calculating the reward, keep the result non-negative within the output value range. Don't return an exact zero (0) value.

Note

AWS DeepRacer supports the `math` package in the `reward_function` by default.

Simple Reward Function Examples

We can start building the reward function by first considering the most basic situation. The situation is driving on a straight track from start to finish without going off the track. In this scenario, the reward function logic depends only on `on_track` and `progress`. As a trial, you could start with the following logic:

```
def reward_function(params):
    if not params["all_wheels_on_track"]:
        reward = -1
    else if params["progress"] == 1 :
        reward = 10
    return reward
```

This logic penalizes the agent when it drives itself off the track. It rewards the agent when it drives to the finishing line. It's reasonable for achieving the stated goal. However, the agent roams freely between the starting point and the finishing line, including driving backwards on the track. Not only could the training take a long time to complete, but also the trained model would lead to less efficient driving when deployed to a real-world vehicle.

In practice, an agent learns more effectively if it can do so bit-by-bit throughout the course of training. This implies that a reward function should give out smaller rewards step by step along the track. For the agent to drive on the straight track, we can improve the reward function as follows:

```
def reward_function(params):
    if not params["all_wheels_on_track"]:
```

```
        reward = -1
    else:
        reward = params["progress"]
    return reward
```

With this function, the agent gets more reward the closer it reaches the finishing line. This should reduce or eliminate unproductive trials of driving backwards. In general, we want the reward function to distribute the reward more evenly over the action space. Creating an effective reward function can be a challenging undertaking. You should start with a simple one and progressively enhance or improve the function. With systematic experimentation, the function can become more robust and efficient.

Enhance Your Reward Function

After you have successfully trained your AWS DeepRacer model for the simple straight track, the AWS DeepRacer vehicle (virtual or physical) can drive itself without going off the track. If you let the vehicle run on a looped track, it won't stay on the track. The reward function has ignored the actions to make turns to follow the track.

To make your vehicle handle those actions, you must enhance the reward function. The function must give a reward when the agent makes a permissible turn and produce a penalty if the agent makes an illegal turn. Then, you're ready to start another round of training. To take advantage of the prior training, you can start the new training by cloning the previous trained model, passing along the previously learned knowledge. You can follow this pattern to gradually add more features to the reward function to train your AWS DeepRacer vehicle to drive in increasingly more complex environments.

To illustrate how the reward function influences desired behaviors on your agent, we show in this section how to iteratively enhance and improve a reward function with a series of example functions that take the [input parameters as described elsewhere \(p. 57\)](#):

Topics

- [Example 1: Follow the Track Center \(p. 31\)](#)
- [Example 2: Stay Inside Two Track Borders \(p. 32\)](#)
- [Example 3: Drive Without Excessive Zig-zagging \(p. 32\)](#)

Example 1: Follow the Track Center

In the following example, we start with a reward function to keep the vehicle driving close to the center of a track.

```
def reward_function(params):
    """
    Example of rewarding the agent to follow the track center line
    """

    # Read input parameters
    track_width = params['track_width']
    distance_from_center = abs(params['distance_from_center'])

    # Calculate 3 marks that are farther and father 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
    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 reward

```

This example reward function determines how far away the agent is from the center line. It then progressively assigns a higher reward as the agent gets closer to the center of the track in discrete steps.

Example 2: Stay Inside Two Track Borders

In the following example function, we add rewards or penalties to steering to prevent the vehicle from turning away from the center line of the track.

```

def reward_function(params):
    """
    Example of rewarding the agent to stay inside the two track borders
    """

    # Read input parameters
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = abs(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 vehicle 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 reward

```

This example reward function simply gives high rewards if the agent stays inside the borders. Otherwise, it lets the agent roam free to find the best path to finish the course. It is straightforward to program and understand, but the training is likely to take longer time to converge.

Example 3: Drive Without Excessive Zig-zagging

In this example, we add rewards or penalties to vehicle's orientation to keep the vehicle's body straight while it drives along the track center.

```

def reward_function(params):
    """
    Example of penalize steering, which helps mitigate zig-zag behaviors
    """

    # Read input parameters
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    steering = abs(params['steering_angle']) # Only need the absolute steering angle

    # Calculate 3 marks that are farther and father 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
    else:
        reward = -(distance_from_center / marker_1)

```

```

        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

        # Steering penalty threshold, change the number based on your action space setting
        ABS_STEERING_THRESHOLD = 15

        # Penalize reward if the car is steering too much
        if steering > ABS_STEERING_THRESHOLD:
            reward *= 0.8

    return float(reward)

```

This example reward function encourages the agent with higher reward to follow the track center line but penalizes it with lower reward when the agent steers too much. The effect is to help prevent zig-zag driving behaviors. The agent learns to drive smoothly in the simulator and likely to keep the same behavior when deployed in the physical vehicle.

Explore Action Space to Train a Robust Model

As a general rule, train your model to be as robust as possible so that you can apply it to as many environments as possible. A robust model is one that can be applied to a wide range of track shapes and conditions. Generally speaking, a robust model is not "smart" because its reward function does not have the ability to contain explicit environment-specific knowledge. Otherwise, your model is likely to be applicable only to an environment similar to the trained one.

Explicitly incorporating environment-specific information into the reward function amounts to feature engineering. Feature engineering helps reduce training time and can be useful in solutions tailor made to a particular environment. To train a model of the general applicability though, you should refrain from attempting a lot of feature engineering.

For example, when training a model on a circular track, you can't expect to obtain a trained model applicable to any non-circular track if you have such geometric properties explicitly incorporated into the reward function.

How would you go about training a model as robust as possible while keeping the reward function as simple as possible? One way is to explore the action space spanning the actions your agent can take. Another is to experiment with [hyperparameters \(p. 34\)](#) of underlying training algorithm. Often times, you do both. Here, we focus on how to explore the action space to train a robust model for your AWS DeepRacer vehicle.

In training an AWS DeepRacer model, an action (a) is a combination of speed (t meters per second) and steering angle (s in degrees). The action space of the agent defines the ranges of speed and steering angle the agent can take. For a discrete action space of m number of speeds, (v_1, \dots, v_m) and n number of steering angles, (s_1, \dots, s_n), there are $m*n$ possible actions in the action space:

$a_1:$	(v_1, s_1)
\dots	
$a_n:$	(v_1, s_n)
\dots	
$a_{(m-1)*n+j}:$	(v_i, s_j)
\dots	
$a_{m*n}:$	(v_m, s_1)
\dots	
$a_{m*n}:$	(v_m, s_n)

The actual values of (v_i , s_j) depend on the ranges of v_{\max} and $|s_{\max}|$ and are not uniformly distributed.

Each time you begin training or iterating your AWS DeepRacer model, you must first specify the n , m , v_{\max} and $|s_{\max}|$ or agree to using their default values. Based on your choice, the AWS DeepRacer service generates the available actions your agent can choose in training. The generated actions are not uniformly distributed over the action space.

In general, a larger number of actions and larger action ranges give your agent more room or options to react to more varied track conditions, such as a curved track with irregular turning angles or directions. The more options available to the agent, the more readily it can handle track variations. As a result, you can expect that the trained model to be more widely applicable, even when using a simple reward function .

For example, your agent can learn quickly to handle straight-line track using a coarse-grained action space with small number of speeds and steering angles. On a curved track, this coarse-grained action space is likely to cause the agent to overshoot and go off the track while it turns. This is because there are not enough options at its disposal in order to adjust its speed or steering. Increase the number of speeds or the number of steering angles or both, the agent should become more capable of maneuvering the curves while keeping on the track. Similarly, if your agent moves in a zig-zag fashion, you can try to increase the number of steering ranges to reduce drastic turns at any given step.

When the action space is too large, training performance may suffer, because it takes longer to explore the action space. Be sure to balance the benefits of a model's general applicability against its training performance requirements. This optimization involves systematic experimentation.

Systematically Tune Hyperparameters

One way to improve your model's performance is to enact a better or more effective training process. For example, to obtain a robust model, training must provide your agent more or less evenly distributed sampling over the agent's action space. This requires a sufficient mix of exploration and exploitation. Variables affecting this include the amount of training data used (number of episodes between each training and batch size), how fast the agent can learn (learning rate), the portion of exploration (entropy). To make training practical, you may want to speed the learning process. Variables affecting this include learning rate, batch size, number of epochs and discount factor.

The variables affecting the training process are known as hyperparameters of the training. These algorithm attributes are not properties of the underlying model. Unfortunately, hyperparameters are empirical in nature. Their optimal values are not known for all practical purposes and require systematic experimentation to derive.

Before discussing the hyperparameters that can be adjusted to tune the performance of training your AWS DeepRacer model, let's define the following terminology.

Data point

A data point, also known as an *experience*, it is a tuple of (s, a, r, s') , where s stands for an observation (or state) captured by the camera, a for an action taken by the vehicle, r for the expected reward incurred by the said action, and s' for the new observation after the action is taken.

Episode

An episode is a period in which the vehicle starts from a given starting point and ends up completing the track or going off the track. It embodies a sequence of experiences. Different episodes can have different lengths.

Experience buffer

An experience buffer consists of a number of ordered data points collected over fixed number of episodes of varying lengths during training. For AWS DeepRacer, it corresponds to images captured by the camera mounted on your AWS DeepRacer vehicle and actions taken by the vehicle and serves

as the source from which input is drawn for updating the underlying (policy and value) neural networks.

Batch

A batch is an ordered list of experiences, representing a portion of simulation over a period of time, used to update the policy network weights. It is a subset of the experience buffer.

Training data

A training data is a set of batches sampled at random from an experience buffer and used for training the policy network weights.

Algorithmic hyperparameters and their effects

Hyperparameter	Description
Gradient descent batch size	<p>The number recent vehicle experiences sampled at random from an experience buffer and used for updating the underlying deep-learning neural network weights. Random sampling helps reduce correlations inherent in the input data. Use a larger batch size to promote more stable and smooth updates to the neural network weights, but be aware of the possibility that the training may be longer or slower.</p> <p>Required</p> <p>Yes</p> <p>Valid values</p> <p>Positive integer of (32, 64, 128, 256, 512)</p> <p>Default value</p> <p>64</p>
Number of epochs	<p>The number of passes through the training data to update the neural network weights during gradient descent. The training data corresponds to random samples from the experience buffer. Use a larger number of epochs to promote more stable updates, but expect a slower training. When the batch size is small, you can use a smaller number of epochs</p> <p>Required</p> <p>No</p> <p>Valid values</p> <p>Positive integer between [3 – 10]</p> <p>Default value</p> <p>3</p>
Learning rate	<p>During each update, a portion of the new weight can be from the gradient-descent (or ascent) contribution and the rest from the existing weight value. The learning rate controls how much a gradient-descent (or ascent) update contributes to the network weights. Use a higher learning rate to include more gradient-descent contributions for faster training, but be aware of the possibility that the expected reward may not converge if the learning rate is too large.</p> <p>Required</p> <p>No</p>

Hyperparameter	Description
	<p>Valid values</p> <p>Real number between 0.00000001 (or 10^{-8}) and 0.001 (or 10^{-3})</p> <p>Default value</p> <p>0.0003</p>
Entropy	<p>A degree of uncertainty used to determine when to add randomness to the policy distribution. The added uncertainty helps the AWS DeepRacer vehicle explore the action space more broadly. A larger entropy value encourages the vehicle to explore the action space more thoroughly.</p> <p>Required</p> <p>No</p> <p>Valid values</p> <p>Real number between 0 and 1.</p> <p>Default value</p> <p>0.01</p>
Discount factor	<p>A factor specifies how much of the future rewards contribute to the expected reward. The larger the Discount factor value is, the farther out contributions the vehicle considers to make a move and the slower the training. With the discount factor of 0.9, the vehicle includes rewards from an order of 10 future steps to make a move. With the discount factor of 0.999, the vehicle considers rewards from an order of 1000 future steps to make a move. The recommended discount factor values are 0.99, 0.999 and 0.9999.</p> <p>Required</p> <p>No</p> <p>Valid values</p> <p>Real number between 0 and 1.</p> <p>Default value</p> <p>0.999</p>

Hyperparameter	Description
Loss type	<p>Type of the objective function used to update the network weights. A good training algorithm should make incremental changes to the agent's strategy so that it gradually transitions from taking random actions to taking strategic actions to increase reward. But if it makes too big a change then the training becomes unstable and the agent ends up not learning. The Huber loss and Mean squared error loss types behave similarly for small updates. But as the updates become larger, Huber loss takes smaller increments compared to Mean squared error loss. When you have convergence problems, use the Huber loss type. When convergence is good and you want to train faster, use the Mean squared error loss type.</p> <p>Required</p> <p>No</p> <p>Valid values</p> <p>(Huber loss, Mean squared error loss)</p> <p>Default value</p> <p>Huber loss</p>
Number of experience episodes between each policy-updating iteration	<p>The size of the experience buffer used to draw training data from for learning policy network weights. An experience episode is a period in which the agent starts from a given starting point and ends up completing the track or going off the track. It consists of a sequence of experiences. Different episodes can have different lengths. For simple reinforcement-learning problems, a small experience buffer may be sufficient and learning is fast. For more complex problems that have more local maxima, a larger experience buffer is necessary to provide more uncorrelated data points. In this case, training is slower but more stable. The recommended values are 10, 20 and 40.</p> <p>Required</p> <p>No</p> <p>Valid values</p> <p>Integer between 5 and 100</p> <p>Default value</p> <p>20</p>

Clone a Trained Model to Start a New Training Pass

If you clone a previously trained model as the starting point of a new round of training, you could improve training efficiency. To do this, modify the hyperparameters to make use of already learned knowledge.

In this section, you learn how to clone a trained model using the AWS DeepRacer console.

To iterate training the reinforcement learning model using the AWS DeepRacer console

1. Sign in to the AWS DeepRacer console, if you're not already signed in.
2. On the **Models** page, choose a trained model and then choose **Clone** from the **Action** drop-down menu list.
3. For **Model details**, do the following:

- a. Type `RL_model_1` in **Model name**, if you don't want a name to be generated for the cloned model.
 - b. Optionally, give a description for the to-be-cloned model in **Model description - optional**.
4. For **Environment simulation**, choose another track option.
 5. For **Reward function**, choose one of the available reward function examples. Modify the reward function. For example, consider steering.
 6. Expand **Algorithm settings** and try different options. For example, change the **Gradient descent batch size** value from 32 to 64 or increase the **Learning rate** to speed up the training.
 7. Experiment with difference choices of the **Stop conditions**.
 8. Choose **Start training** to begin new round of training.

As with training a robust machine learning model in general, it is important that you conduct systematic experimentation to come up with the best solution.

Evaluate AWS DeepRacer Models in Simulations

To evaluate a model is to test the performance of a trained model. In AWS DeepRacer, the standard performance metric is the average time of finishing three consecutive laps. Using this metric, of any two models, one is better if it can make the agent go faster on the same track than the other one.

In general, evaluating a model involves the following tasks:

1. Configure and start an evaluation job.
2. Observe the evaluation in progress while the job is running. This can be done in the AWS DeepRacer simulator.
3. Inspect the evaluation summary after the evaluation job is done. You can terminate an evaluation job in progress at any time.
4. Optionally, submit the evaluation result to an eligible [AWS DeepRacer leaderboard \(p. 109\)](#). The ranking on the leaderboard lets you know how well your model performs against other participants.

For step-by-step instructions to run an AWS DeepRacer evaluation job in simulation, see [the section called "Evaluate Models in Simulation" \(p. 14\)](#). You can run multiple evaluation jobs in simulation, but you must run them one after another. AWS DeepRacer only keeps the latest evaluation job status and the result.

Test an AWS DeepRacer model with an AWS DeepRacer vehicle driving on a physical track, see [Operate Your Vehicle \(p. 70\)](#).

Log AWS DeepRacer Events to CloudWatch Logs

For diagnostic purposes, AWS DeepRacer reports certain runtime events to CloudWatch Logs during training and evaluation.

The events are logged in job-specific log streams. For a training job, the log stream appears under the `/aws/sagemaker/TrainingJobs` log group. For a simulation job, the log stream appears under the `/aws/robomaker/SimulationJobs` log group. For an evaluation job submitted to a leaderboard in the DeepRacer League Virtual circuit, the log stream appears under the `/aws/deepracer/leaderboard/SimulationJobs` log group. For the reward function execution, the log stream appears under the `/aws/lambda/AWS-DeepRacer-Test-Reward-Function` log group.

Most of the log entries are self-explanatory, except for those starting with `"SIM_TRACE_LOG"`. An example of this log entry is shown as follows:

```
SIM_TRACE_LOG:0,14,3.1729,0.6200,-0.2606,-0.26,0.50,2,0.5000,False,True,1.4878,1,17.67,1563406790.24001
```

The event items correspond to the following data values, respectively:

```
SIM_TRACE_LOG: episode, step, x-coordinate, y-
coordinate, heading, steering_angle, speed, action_taken, reward, job_completed, all_wheels_on_track, p
closest_waypoint_index, track_length, time.time()
```

To access the AWS DeepRacer logs, you can use the [CloudWatch](#) console , the AWS CLI or an AWS SDK.

To view AWS DeepRacer logs using the AWS CLI

1. Open a terminal window.
2. Type the following command:

```
aws logs get-log-events \
--log-group-name a-deepracer-log-group-name \
--log-stream-name a-deepracer-log-stream-name
```

The command returns a result similar to the following output:

```
{
  "events": [
    {
      "timestamp": 1563406819300,
      "message":
"SIM_TRACE_LOG:2,155,7.3941,1.0048,0.0182,-0.52,1.00,1,0.0010,False,False,14.7310,16,1
7.67,1563406818.939216",
      "ingestionTime": 1563406819310
    },
    ...
    {
      "timestamp": 1563407217100,
      "message":
"SIM_TRACE_LOG:39,218,5.6879,0.3078,-0.1135,0.52,1.00,9,0.0000,True,False,20.7185,9,17.67,15634072
17108",
      "ingestionTime": 1563407217108
    },
    {
      "timestamp": 1563407218143,
      "message": "Training> Name=main_level/agent, Worker=0, Episode=40, Total
reward=61.93, Steps=4315, Training iteration=0",
      "ingestionTime": 1563407218150
    }
  ],
  "nextForwardToken": "f/34865146013350625778794700014105997464971505654143647744",
  "nextBackwardToken": "b/34865137118854508561245373892407536877673471318173089813"
}
```

To view AWS DeepRacer logs in the CloudWatch Logs console:

1. Sign in to the [CloudWatch](#) console.
2. Choose **Logs** from the main navigation pane.
3. Choose an appropriate log group.

To help quickly find the AWS DeepRacer-specific event logs, type one of the aforementioned log group names in the **Filter** box.

4. Choose a log stream to open the log file.

To quickly locate the most recent log stream in a given log group, sort the list by **Last Event Time**.

Optimize Training AWS DeepRacer Models for Real Environments

Many factors affect the real-world performance of a trained model, including the choice of the [action space \(p. 33\)](#), [reward function \(p. 29\)](#), [hyperparameters \(p. 34\)](#) used in the training, and [vehicle calibration \(p. 85\)](#) as well as [real-world track \(p. 100\)](#) conditions. In addition, the simulation is only an (often crude) approximation of the real world. They make it a challenge to train a model in simulation, to apply it to the real world, and to achieve a satisfactory performance.

Training a model to give a solid real-world performance often requires numerous iterations of exploring the [reward function \(p. 29\)](#), [action spaces \(p. 33\)](#), [hyperparameters \(p. 34\)](#), and [evaluation \(p. 38\)](#) in simulation and [testing \(p. 92\)](#) in a real environment. The last step involves the so-called *simulation-to-real world (sim2real)* transfer and can feel unwieldy.

To help tackle the *sim2real* challenges, heed the following considerations:

- Make sure that your vehicle is well calibrated.

This is important because the simulated environment is most likely a partial representation of the real environment. Besides, the agent takes an action based on the current track condition, as captured by an image from the camera, at each step. It cannot see far enough to plan its route at a fast speed. To accommodate this, the simulation imposes limits on the speed and steering. To ensure the trained model works in the real world, the vehicle must be properly calibrated to match this and other simulation settings. For more information for calibrating your vehicle, see [the section called “Calibrate Your Vehicle” \(p. 85\)](#).

- Test your vehicle with the default model first.

Your AWS DeepRacer vehicle comes with a pre-trained model loaded into its inference engine. Before testing your own model in the real world, verify that the vehicle performs reasonably well with the default model. If not, check the physical track setup. Testing a model in an incorrectly built physical track is likely to lead to a poor performance. In such cases, reconfigure or repair your track before starting or resuming testing.

Note

When running your AWS DeepRacer vehicle, actions are inferred according to the trained policy network without invoking the reward function.

- Make sure the model works in simulation.

If your model doesn't work well in the real world, it's possible that either the model or track is defective. To sort out the root causes, you should first [evaluate the model in simulations \(p. 38\)](#) to check if the simulated agent can finish at least one loop without getting off the track. You can do so by inspecting the convergence of the rewards while observing the agent's trajectory in the simulator. If the reward reaches the maximum when the simulated agents completes a loop without faltering, the model is likely to be a good one.

- Do not over train the model.

Continuing training after the model has consistently completed the track in simulation will cause overfitting in the model. An over-trained model won't perform well in the real world because it can't handle even minor variations between the simulated track and the real environment.

- Use multiple models from different iterations.

A typical training session produces a range of models that fall between being underfitted and being overfitted. Because there are no a priori criteria to determine a model that is just right, you should pick a few model candidates from the time when the agent completes a single loop in the simulator to the point where it performs loops consistently.

- Start slow and increase the driving speed gradually in testing.

When testing the model deployed to your vehicle, start with a small maximum speed value. For example, you can set the testing speed limit to be <10% of the training speed limit. Then gradually increase the testing speed limit until the vehicle starts moving. You set the testing speed limit when calibrating the vehicle using the device control console. If the vehicle goes too fast, i.e. the speed exceeds those seen during training in simulator, the model is not likely to perform well on the real track.

- Test a model with your vehicle in different starting positions.

The model learns to take a certain path in simulation and can be sensitive to its position within the track. You should start the vehicle tests with different positions within the track boundaries (from left to center to right) to see if the model performs well from certain positions. Most models tend to make the vehicle stay close to either side of one of the white lines. To help analyze the vehicle's path, plot the vehicle's positions (x, y) step by step from the simulation to identify likely paths to be taken by your vehicle in a real environment.

- Start testing with a straight track.

A straight track is much easier to navigate compared to a curved track. Starting your test with a straight track is useful to weed out poor models quickly. If a vehicle cannot follow a straight track most of the time, the model will not perform well on curved tracks, either.

- Watch out for the behavior where the vehicle takes only one type of actions,

When your vehicle can manage to take only one type of actions, e.g., to steer the vehicle to the left only, the model is likely over-fitted or under-fitted. With given model parameters, too many iterations in training could make the model over-fitted. Too few iterations could make it under-fitted.

- Watch out for vehicle's ability to correct its path along a track border.

A good model makes the vehicle to correct itself when nearing the track borders. Most well-trained models have this capability. If the vehicle can correct itself on both the track borders, the model is considered to be more robust and of a higher quality.

- Watch out for inconsistent behaviors exhibited by the vehicle.

A policy model represents a probability distribution for taking an action in a given state. With the trained model loaded to its inference engine, a vehicle will pick the most probable action, one step at time, according to the model's prescription. If the action probabilities are evenly distributed, the vehicle can take any of the actions of the equal or closely similar probabilities. This will lead to an erratic driving behavior. For example, when the vehicle follows a straight path sometimes (e.g., half the time) and makes unnecessary turns at other times, the model is either under-fitted or over-fitted.

- Watch out for only one type of turns (left or right) made by the vehicle.

If the vehicle takes left turns very well but fails to manage steering right, or, similarly, if the vehicle takes only right turns well, but not left steering, you need to carefully calibrate or recalibrate your vehicle's steering. Alternatively, you can try to use a model that is trained with the settings close to the physical settings under testing.

- Watch out for the vehicle's making sudden turns and go off-track.

If the vehicle follows the path correctly most of the way, but suddenly veers off the track, it is likely due to distractions in the environment. Most common distractions include unexpected or unintended light reflections. In such cases, use barriers around the track or other means to reduce glaring lights.

Train and Evaluate AWS DeepRacer Models Using Amazon SageMaker Notebooks

The AWS DeepRacer console provides you with an integrated experience to train and evaluate your AWS DeepRacer models. It's integrated because AWS DeepRacer uses Amazon SageMaker and AWS RoboMaker behind the scenes. The integration includes detailed reinforcement learning tasks and makes training more readily accessible to beginners.

If you're an experienced user of Amazon SageMaker or if you're determined to learn how to use Amazon SageMaker and AWS RoboMaker to train and evaluate your AWS DeepRacer models, then you can manually create an Amazon SageMaker notebook. You can then clone a reinforcement learning sample notebook instance and use it as a template to perform the predefined tasks that train and evaluate an AWS DeepRacer model.

After the training, you can copy the trained model artifacts to your AWS DeepRacer vehicle for test runs in a physical environment.

The tutorial presents step-by-step instructions to walk you through these tasks.

Topics

- [Create an Amazon SageMaker Notebook \(p. 42\)](#)
- [Initialize the Amazon SageMaker Notebook Instance \(p. 43\)](#)
- [Set Up the Training Environment \(p. 47\)](#)
- [Train Your AWS DeepRacer Model \(p. 50\)](#)

Create an Amazon SageMaker Notebook

To train an AWS DeepRacer model directly on Amazon SageMaker, follow the steps below and create an Amazon SageMaker notebook instance.

To create an Amazon SageMaker notebook instance to train and evaluate your AWS DeepRacer models

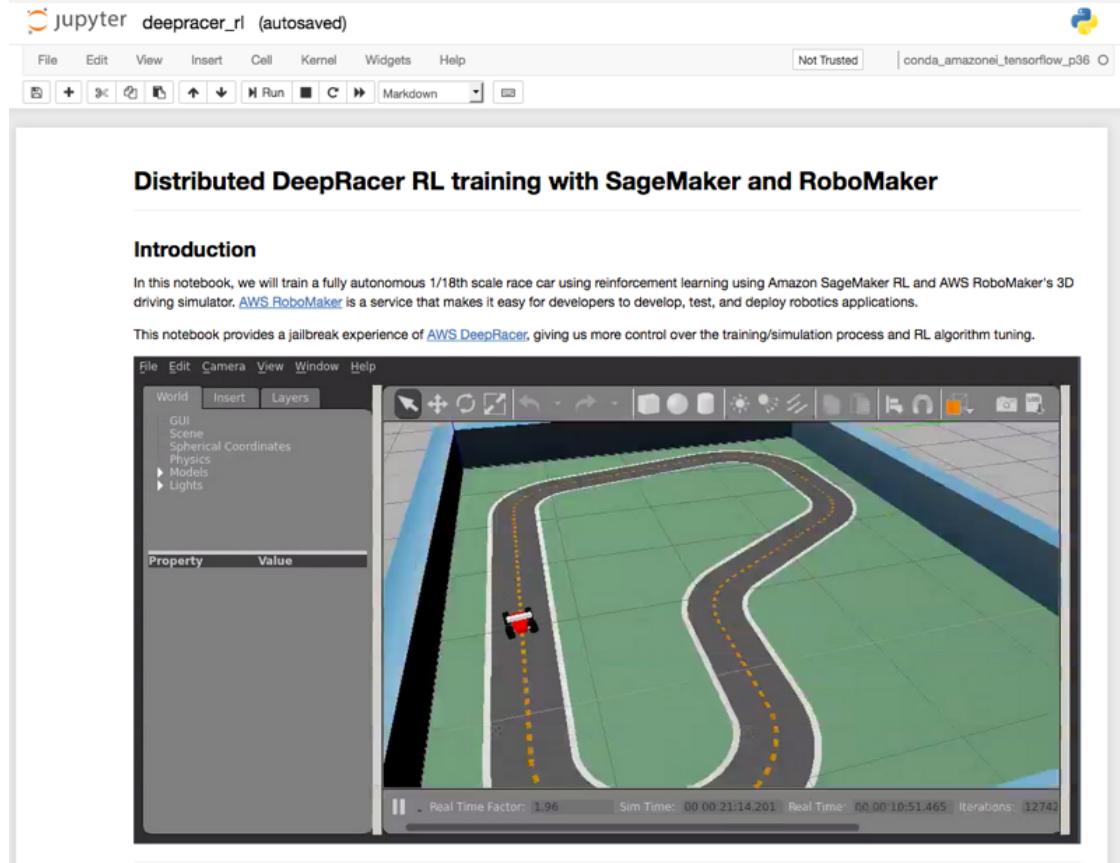
1. Sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>. Choose one of the supported regions.
2. From the navigation pane, choose **Notebook instances** and then choose **Create notebook instance**.
3. On the **Create notebook instance** page, do the following:
 - a. Type a name. For example, `my-deepracer-model`) for the **Notebook instance name**.
 - b. If the **IAM role** drop-down menu is not populated with an existing IAM role, choose **Create a new role**, **Enter a custom IAM role ARN**, or **Use existing role** and then follow the instructions.
 - c. Leave the default choices for all other options and then choose **Create notebook instance**.

For more information, see [creating an Amazon SageMaker notebook instance](#).

4. Wait for the notebook instance's **Status** to change from **Pending** to **InService**. Then choose **Open Jupyter**.
5. On the **Jupyter** page (which is the home page of the newly created notebook), do the following:
 - a. Choose the **SageMaker Examples** tab.
 - b. Expand the **Reinforcement Learning** example group from the example collection.
 - c. For this exercise, choose **Use** next to the `deepracer_rl.ipynb` item.

- d. On the **Create a copy in your home directory** dialog, choose **Create copy**.

At this point, the notebook instance is running and you can begin to train the model.



You are charged for a running instance according to the selected instance type. To avoid being charged for a running instance when you're not ready to use it, shut down the instance.

Initialize the Amazon SageMaker Notebook Instance

To use an Amazon SageMaker notebook instance to train your AWS DeepRacer model, first properly initialize the instance for the required job. The initialization includes the following.

- Import required libraries.
- Set up training environment.
- Grant access permissions for Amazon SageMaker and AWS RoboMaker.
- Provision a docker container to host training and evaluation jobs.
- Configure VPC for Amazon SageMaker and AWS RoboMaker to interact with each other.

Follow the steps below for detailed instructions to initialize a notebook instance.

To initialize an Amazon SageMaker notebook instance

1. To import the required library to do training, choose the notebook instance's first code block. For example, choose the one under the **Imports** heading. Next, choose **Run** from the notebook's menu

bar to execute the code block. You can use the Shift+Enter key-command shortcuts to start running the code block.

```
In [ ]: import boto3
import sagemaker
import sys
import os
import re
import numpy as np
import subprocess
sys.path.append("common")
from misc import get_execution_role, wait_for_s3_object
from docker_utils import build_and_push_docker_image
from sagemaker.rl import RLEstimator, RLToolkit, RLFramework
from time import gmtime, strftime
import time
from IPython.display import Markdown
from markdown_helper import *
```

Before the code execution starts, the code block status shows In []. When the execution is under way, the status becomes In [*]. After the code execution is complete, the status becomes In [n], where n corresponds to the order of invocations. Because the importation code cell is the first, n=1. If you run the command again after the first run, the status becomes In [2].

For asynchronous execution, the code cell returns immediately to show the completed status. For synchronous executions, subsequent calls are blocked until the current code cell execution is completed when the status turns from In [*] to In [n].

2. To initialize the basic parameters, run the **Initializing basic parameters** code block as-is.

Initializing basic parameters

```
In [ ]: # Select the instance type
instance_type = "ml.c4.2xlarge"
#instance_type = "ml.p2.xlarge"
#instance_type = "ml.c5.4xlarge"

# Starting SageMaker session
sage_session = sagemaker.session.Session()

# Create unique job name.
job_name_prefix = 'depracer-notebook'

# Duration of job in seconds (1 hours)
job_duration_in_seconds = 3600

# AWS Region
aws_region = sage_session.boto_region_name
if aws_region not in ["us-west-2", "us-east-1", "eu-west-1"]:
    raise Exception("This notebook uses RoboMaker which is available only in US East (N. Virginia),"
                    "US West (Oregon) and EU (Ireland). Please switch to one of these regions.")
```

The example notebook instance sets the job duration for 1 hour by default. To speed up or extend the training, you can decrease or increase the job_duration_in_seconds value before running the code cell.

3. To set up the training output storage, choose the code block under **Setup S3 bucket**, and then choose **Run** from the notebook instance menu or press the Shift+Enter keys.

Setup S3 bucket

Set up the linkage and authentication to the S3 bucket that we want to use for checkpoint and metadata.

```
In [ ]: # S3 bucket
s3_bucket = sage_session.default_bucket()

# SDK appends the job name and output folder
s3_output_path = 's3://{}{}'.format(s3_bucket)

#Ensure that the S3 prefix contains the keyword 'sagemaker'
s3_prefix = job_name_prefix + "-sagemaker-" + strftime("%Y%m%d-%H%M%S", gmtime())

# Get the AWS account id of this account
sts = boto3.client("sts")
account_id = sts.get_caller_identity()['Account']

print("Using s3 bucket {}".format(s3_bucket))
print("Model checkpoints and other metadata will be stored at: \ns3://{}{}".format(s3_bucket, s3_prefix))
```

When the execution completes, you can verify this bucket in Amazon S3 console.

To view the s3_output_path variable value, append `print(s3_output_path)` to the above code cell and rerun the code.

4. To set up appropriate permissions for this notebook instance to access the S3 storage for output by Amazon SageMaker, run the code cell under **Create an IAM role**.

Create an IAM role

Either get the execution role when running from a SageMaker notebook `role = sagemaker.get_execution_role()` or, when running from local machine, use utils method `role = get_execution_role('role_name')` to create an execution role.

```
In [ ]: try:
    sagemaker_role = sagemaker.get_execution_role()
except:
    sagemaker_role = get_execution_role('sagemaker')

print("Using Sagemaker IAM role arn: \n{}".format(sagemaker_role))
```

When executed, this code block creates a new IAM role containing the following IAM policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3>ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::/*"
      ]
    }
  ]
}
```

The created IAM role has Amazon SageMaker as its trusted entity.

5. To set up appropriate permissions for this notebook instance to invoke AWS RoboMaker to simulate the training environment, run the code cell under **Permission setup for invoking AWS RoboMaker from this notebook** and follow the instructions thereafter to add `robomaker.amazonaws.com` as another trusted entity of the previously created IAM role.

Permission setup for invoking AWS RoboMaker from this notebook

In order to enable this notebook to be able to execute AWS RoboMaker jobs, we need to add one trust relationship to the default execution role of this notebook.

```
In [5]: display(Markdown(generate_help_for_robomaker_trust_relationship(sagemaker_role)))
```

1. Go to IAM console to edit current SageMaker role: [AmazonSageMaker-ExecutionRole-20190515T141689](#).
2. Next, go to the Trust relationships tab and click on Edit Trust Relationship.
3. Replace the JSON blob with the following:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "sagemaker.amazonaws.com",  
                    "robomaker.amazonaws.com"  
                ]  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

4. Once this is complete, click on Update Trust Policy and you are done.

6. To set up the required permissions for Amazon SageMaker to access the S3 storage, run the code cell under **Permission setup for SageMaker to S3 bucket** and follow the instructions thereafter to attach the **AmazonS3FullAccess** policy to the IAM role previously created.

Permission setup for Sagemaker to S3 bucket

The sagemaker writes the Redis IP address, models to the S3 bucket. This requires PutObject permission on the bucket. Make sure the sagemaker role you are using has this permissions.

```
In [6]: display(Markdown(generate_s3_write_permission_for_sagemaker_role(sagemaker_role)))
```

1. Go to IAM console to edit current SageMaker role: [AmazonSageMaker-ExecutionRole-20190515T141689](#).
2. Next, go to the Permissions tab and click on Attach Policy.
3. Search and select **AmazonS3FullAccess** policy

7. To provision a docker container for running our training and evalution jobs, run the code cell under **Build and push docker image**.

Build and push docker image

The file ./Dockerfile contains all the packages that are installed into the docker. Instead of using the default sagemaker container. We will be using this docker container.

```
In [*]: %%time  
cpu_or_gpu = 'gpu' if instance_type.startswith('ml.p') else 'cpu'  
repository_short_name = "sagemaker-docker-%s" % cpu_or_gpu  
docker_build_args = {  
    'CPU_OR_GPU': cpu_or_gpu,  
    'AWS_REGION': boto3.Session().region_name,  
}  
custom_image_name = build_and_push_docker_image(repository_short_name, build_args=docker_build_args)  
print("Using ECR image %s" % custom_image_name)  
  
Building docker image sagemaker-docker-cpu from Dockerfile  
$ docker build -t sagemaker-docker-cpu -f Dockerfile . --build-arg CPU_OR_GPU=cpu --build-arg AWS_REGION=us-west-2  
Sending build context to Docker daemon 600.6kB  
Step 1/18 : FROM ubuntu:16.04  
16.04: Pulling from library/ubuntu  
f7277927d38a: Pulling fs layer  
8d3eac894db4: Pulling fs layer  
edf72af6d627: Pulling fs layer  
3e4f86211d23: Pulling fs layer  
3e4f86211d23: Waiting  
8d3eac894db4: Verifying Checksum  
8d3eac894db4: Download complete  
edf72af6d627: Verifying Checksum  
edf72af6d627: Download complete  
3e4f86211d23: Verifying Checksum  
3e4f86211d23: Download complete  
f7277927d38a: Verifying Checksum  
f7277927d38a: Download complete  
f7277927d38a: Pull complete  
9d3eac894db4: Pull complete
```

Building and pushing the docker image takes some time to finish.

8. To enable VPC mode for Amazon SageMaker and AWS RoboMaker to communicate with each other over network, run the code cell under **Configure VPC**. By default, the notebook instance uses your default VPC, security group, and subnets to configure the VPC mode. If you don't want open VPC for other traffic, make sure to set the **Inbound Rules** and **Outbound Rules** for the specified security group to allow incoming traffic from itself only.



9. To enable the SageMaker training job to access S3 resources, run the code cell under **Create Route Table** to create a VPC S3 endpoint.

Create Route Table

A SageMaker job running in VPC mode cannot access S3 resources. So, we need to create a VPC S3 endpoint to allow S3 access from SageMaker container. To learn more about the VPC mode, please visit [this link](#).

```

In [9]: #TODO: Explain to customer what CREATE_ROUTE_TABLE is doing
CREATE_ROUTE_TABLE = True

def create_vpc_endpoint_table():
    print("Creating ")
    try:
        route_tables = [route_table["RouteTableId"] for route_table in ec2.describe_route_tables()['RouteTables']]
        if route_table['VpcId'] == deepracer_vpc
    except Exception as e:
        if "UnauthorizedOperation" in str(e):
            display(Markdown(generate_help_for_s3_endpoint_permissions(sagemaker_role)))
        else:
            display(Markdown(create_s3_endpoint_manually(aws_region, deepracer_vpc)))
        raise e

    print("Trying to attach S3 endpoints to the following route tables:", route_tables)

    if not route_tables:
        raise Exception("No route tables were found. Please follow the VPC S3 endpoint creation "
                       "guide by clicking the above link.")
    try:
        ec2.create_vpc_endpoint(DryRun=False,
                               VpcEndpointType="Gateway",
                               VpcId=deepracer_vpc,
                               ServiceName="com.amazonaws.{}.s3".format(aws_region),
                               RouteTableIds=route_tables)
        print("S3 endpoint created successfully!")
    except Exception as e:
        if "RouteAlreadyExists" in str(e):
            print("S3 endpoint already exists.")
        elif "UnauthorizedOperation" in str(e):
            display(Markdown(generate_help_for_s3_endpoint_permissions(role)))
        raise e
        else:
            display(Markdown(create_s3_endpoint_manually(aws_region, default_vpc)))
            raise e

    if CREATE_ROUTE_TABLE:
        create_vpc_endpoint_table()

Creating
Trying to attach S3 endpoints to the following route tables: ['rtb-0f4c136a']
S3 endpoint already exists.

```

At this point, you're done with initializing the training and are ready to move on to [set up the training environment \(p. 47\)](#).

Set Up the Training Environment

Setting up the environment for training your AWS DeepRacer model involves selecting a race track, a reward function and the associated action space, as well as hyperparameters used for training.

The notebook uses the default settings for these. To view the default settings, uncomment relevant parts and then run the code cell under **Configure the preset for RL algorithm**. For example, to view the code listing of the reward function, run the code cell as follows:

Configure the preset for RL algorithm

The parameters that configure the RL training job are defined in `src/markov/presets/`. Using the preset file, you can define agent parameters to select the specific agent algorithm. We suggest using Clipped PPO for this example. You can edit this file to modify algorithm parameters like learning_rate, neural network structure, batch_size, discount factor etc.

```
In [10]: # Uncomment the pygmentize code lines to see the code

# Environmental File
#!/pygmentize src/markov/environments/deepracer_racetrack_env.py

# Reward function
#!/pygmentize src/markov/rewards/default.py

# Action space
#!/pygmentize src/markov/actions/model_metadata_10_state.json

# Preset File
#!/pygmentize src/markov/presets/default.py
#!/pygmentize src/markov/presets/preset_attention_layer.py

def reward_function(params):

    distance_from_center = params['distance_from_center']
    track_width = params['track_width']

    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    reward = 1e-3
    if distance_from_center <= marker_1:
        reward = 1
    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)
```

If you decide to use the default settings, copy the files to the S3 bucket. To modify any of the files, follow the steps below, changing the file name and directory for anything other than the default reward function.

To modify the reward function in the `default.py` file:

1. Choose **File** menu on the top of the notebook instance page and then choose **Open....**

Jupyter deepracer_rl (autosaved)

File Edit View Insert Cell Kernel Widgets Help

New Notebook Open... Make a Copy... Save as... Rename... Save and Checkpoint Revert to Checkpoint Print Preview Download as Trust Notebook Close and Halt

```
rd function
ntize src/markov/rewards/default.py

on space
entize src/markov/actions/model_metadata_10_state.json

et File
entize src/markov/presets/default.py
entize src/markov/presets/preset_attention_layer.py

ward_function(params):

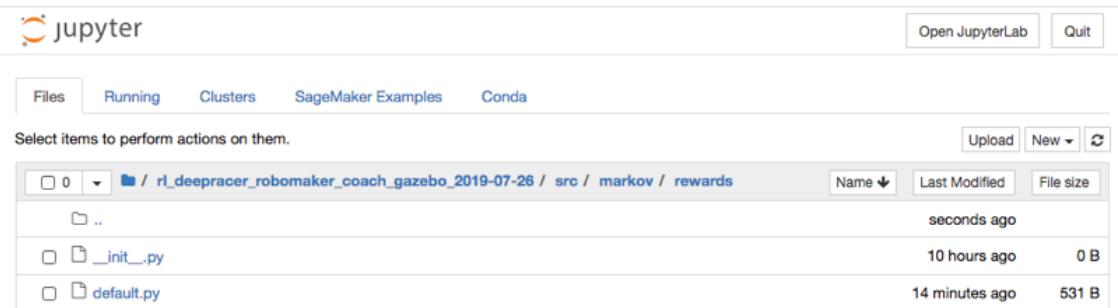
stance_from_center = params['distance_from_center']
ack_width = params['track_width']

rker_1 = 0.1 * track_width
rker_2 = 0.25 * track_width
rker_3 = 0.5 * track_width

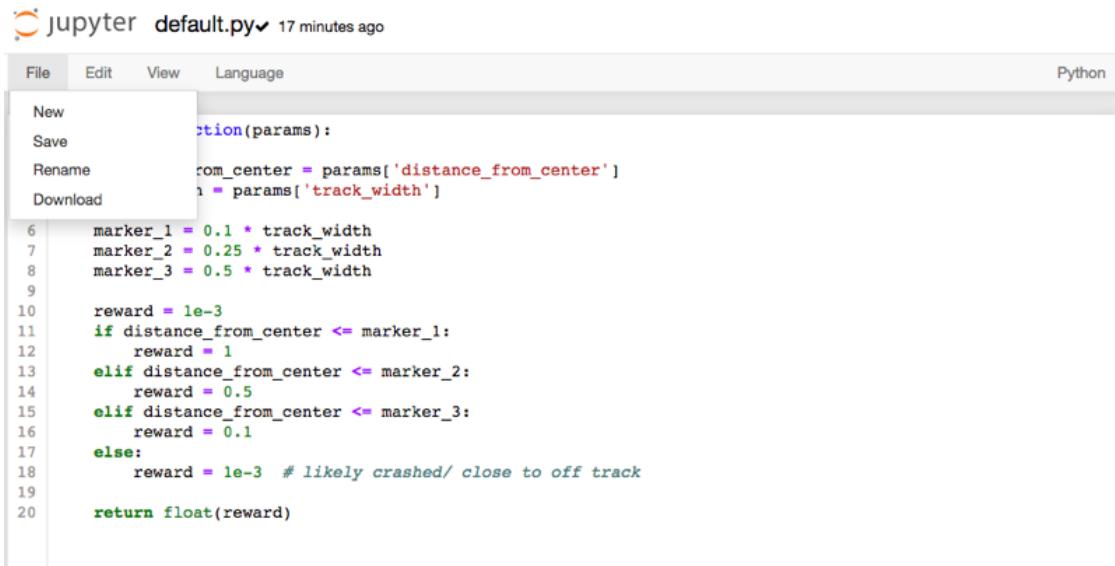
reward = 1e-3
if distance_from_center <= marker_1:
    reward = 1
elif distance_from_center <= marker_2:
    reward = 0.5
elif distance_from_center <= marker_3:
    reward = 0.1
else:
    reward = 1e-3 # likely crashed/ close to off track

return float(reward)
```

2. Navigate to the *src/markov/rewards* folder and choose *default.py* to open the file.



3. Edit the file as you see fit. After finishing editing the file, choose **File->Save** to save the update.



The screenshot shows a Jupyter Notebook interface with a tab labeled "jupyter default.py" and a timestamp "17 minutes ago". The notebook has a "Python" kernel selected. The code in the cell is a Python function named `get_reward` that takes a parameter `distance_from_center` and returns a reward based on the distance from the center track markers. The code uses conditional statements to calculate the reward for different ranges of distance.

```
File Edit View Language Python
New Save Rename Download
1 def get_reward(distance_from_center):
2     # Initialize reward
3     reward = -0.01
4
5     # Define marker positions
6     marker_1 = 0.1 * track_width
7     marker_2 = 0.25 * track_width
8     marker_3 = 0.5 * track_width
9
10    # Calculate reward based on distance from center
11    if distance_from_center <= marker_1:
12        reward = 1
13    elif distance_from_center <= marker_2:
14        reward = 0.5
15    elif distance_from_center <= marker_3:
16        reward = 0.1
17    else:
18        reward = -1e-3 # likely crashed/ close to off track
19
20    return float(reward)
```

Notice that the environment file is shared by both Amazon SageMaker and AWS RoboMaker, also known as nodes. When it's used by Amazon SageMaker, the `node_type` is `SAGEMAKER_TRAINING_WORKER`. When it's used by AWS RoboMaker, the `node_type` is `SIMULATION_WORKER`.

Train Your AWS DeepRacer Model

Training your model with Amazon SageMaker and AWS RoboMaker amounts to executing the code in the `training_worker.py` file under the notebook's `src` directory. The `training_worker.py` file is designated as the entry point of your training job.

The training process involves using AWS RoboMaker to emulate driving experiences in the environment, relaying the experiences at fixed intervals to Amazon SageMaker as input to train the deep neural network, and updating the network weights to an S3 location.

While the training is in progress, you can have specified training metrics logged to Amazon CloudWatch Logs or displayed to the AWS RoboMaker terminal.

To train your AWS DeepRacer model

1. Run the code cell under **Copy custom files to S3 bucket so that sagemaker & robomaker can pick it up** copy the environment files to S3 .

Copy custom files to S3 bucket so that sagemaker & robomaker can pick it up

```
In [11]: s3_location = "s3://%s/%s" % (s3_bucket, s3_prefix)
print(s3_location)

# Clean up the previously uploaded files
!aws s3 rm --recursive {s3_location}

# Make any changes to the environment and preset files below and upload these files
!aws s3 cp src/markov/environments/deepracer_racetrack_env.py {s3_location}/environments/deepracer_racetrack_env.py

!aws s3 cp src/markov/rewards/default.py {s3_location}/rewards/reward_function.py

!aws s3 cp src/markov/actions/model_metadata_10_state.json {s3_location}/model_metadata.json

!aws s3 cp src/markov/presets/default.py {s3_location}/presets/preset.py
#!aws s3 cp src/markov/presets/preset_attention_layer.py {s3_location}/presets/preset.py
```

s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602
upload: src/markov/environments/deepracer_racetrack_env.py to s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602/environments/deepracer_racetrack_env.py
upload: src/markov/rewards/default.py to s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602/rewards/reward_function.py
upload: src/markov/actions/model_metadata_10_state.json to s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602/model_metadata.json
upload: src/markov/presets/default.py to s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602/presets/preset.py

2. To start an Amazon SageMaker job to train your AWS DeepRacer model, do the following:

- a. Run the first code cell under **Train the RL model using the Python SDK Script mode** to define training metrics to watch in either CloudWatch Logs or in an AWS RoboMaker console window.

```
In [60]: metric_definitions = [
    # Training> Name=main_level/agent, Worker=0, Episode=19, Total reward=-102.88, Steps=19019,
    {'Name': 'reward-training',
     'Regex': '^Training>.*Total reward=(.*?),'},
    # Policy training> Surrogate loss=-0.32664725184440613, KL divergence=7.255815035023261e-06
    {'Name': 'ppo-surrogate-loss',
     'Regex': '^Policy training>.*Surrogate loss=(.*?),'},
    {'Name': 'ppo-entropy',
     'Regex': '^Policy training>.*Entropy=(.*?),'},
    # Testing> Name=main_level/agent, Worker=0, Episode=19, Total reward=1359.12, Steps=20015,
    {'Name': 'reward-testing',
     'Regex': '^Testing>.*Total reward=(.*?),'},
]
```

You can watch the specified metrics to monitor the training and to find out the effectiveness of your chosen reward function in CloudWatch Logs or using an AWS RoboMaker terminal.

- b. Run the second code cell under **Train the RL model using the Python SDK Script mode** to start an Amazon SageMaker training job for your model.

We use the RLEstimator for training RL jobs.

1. Specify the source directory which has the environment file, preset and training code.
2. Specify the entry point as the training code
3. Specify the choice of RL toolkit and framework. This automatically resolves to the ECR path for the RL Container.
4. Define the training parameters such as the instance count, instance type, job name, s3_bucket and s3_prefix for storing model checkpoints and metadata. **Only 1 training instance is supported for now.**
5. Set the RLCOACH_PRESET as "deepracer" for this example.
6. Define the metrics definitions that you are interested in capturing in your logs. These can also be visualized in CloudWatch and SageMaker Notebooks.

```
In [21]: estimator = RLEstimator(entry_point="training_worker.py",
                                source_dir='src',
                                image_name=custom_image_name,
                                dependencies=[common],
                                role=sagemaker_role,
                                train_instance_type=instance_type,
                                train_instance_count=1,
                                output_path=s3_output_path,
                                base_job_name=job_name_prefix,
                                metric_definitions=metric_definitions,
                                train_max_run=job_duration_in_seconds,
                                hyperparameters={
                                    "s3_bucket": s3_bucket,
                                    "s3_prefix": s3_prefix,
                                    "aws_region": aws_region,
                                    "preset_s3_key": "%s/presets/preset.py" % s3_prefix,
                                    "model_metadata_s3_key": "%s/model_metadata.json" % s3_prefix,
                                    "environment_s3_key": "%s/environments/deepracer_racetrack_env.py"
                                },
                                subnets=deepracer_subnets,
                                security_group_ids=deepracer_security_groups,
                            )

estimator.fit(wait=False)
job_name = estimator.latest_training_job.job_name
print("Training job: %s" % job_name)
```

Training job: deepracer-notebook-2019-07-26-23-37-45-662

This Amazon SageMaker training job uses the TensorFlow framework and runs on a specified EC2 compute instance type. The output lists the job name. You can track the status of this training job in Amazon SageMaker.

Name	Creation time	Duration	Status
deepracer-notebook-2019-07-26-23-37-45-662	Jul 26, 2019 23:37 UTC	-	InProgress

3. To create an environment simulation job in AWS RoboMaker, run the code cells under **Start the RoboMaker job and Create Simulation Application.**
4. To start the simulation on AWS RoboMaker and share the simulated data, run the code cell under **Launch the Simulation job on RoboMaker.**

Launch the Simulation job on RoboMaker

We create [AWS RoboMaker](#) Simulation Jobs that simulates the environment and shares this data with SageMaker for training.

```
In [26]: num_simulation_workers = 1

envriron_vars = {
    "WORLD_NAME": "reinvent_base",
    "KINESIS_VIDEO_STREAM_NAME": "SilverstoneStream",
    "SAGEMAKER_SHARED_S3_BUCKET": s3_bucket,
    "SAGEMAKER_SHARED_S3_PREFIX": s3_prefix,
    "TRAINING_JOB_ARN": job_name,
    "APP_REGION": aws_region,
    "METRIC_NAME": "TrainingRewardScore",
    "METRIC_NAMESPACE": "AWSDeepRacer",
    "REWARD_FILE_S3_KEY": "%s/rewards/reward_function.py" % s3_prefix,
    "MODEL_METADATA_FILE_S3_KEY": "%s/model_metadata.json" % s3_prefix,
    "METRICS_S3_BUCKET": s3_bucket,
    "METRICS_S3_OBJECT_KEY": s3_bucket + "/training_metrics.json",
    "TARGET_REWARD_SCORE": "None",
    "NUMBER_OF_EPISODES": "0",
    "ROBOMAKER_SIMULATION_JOB_ACCOUNT_ID": account_id
}

simulation_application = {"application": simulation_app_arn,
                        "launchConfig": {"packageName": "deephrcer_simulation_environment",
                                         "launchFile": "distributed_training.launch",
                                         "environmentVariables": envriron_vars}}
}

vpcConfig = {"subnets": deepracer_subnets,
             "securityGroups": deepracer_security_groups,
             "assignPublicIp": True}

client_request_token = strftime("%Y-%m-%d-%H-%M-%S", gmtime())

responses = []
for job_no in range(num_simulation_workers):
    response = robomaker.create_simulation_job(iamRole=sagemaker_role,
                                                clientRequestToken=client_request_token,
                                                maxJobDurationInSeconds=job_duration_in_seconds,
                                                failureBehavior="Continue",
                                                simulationApplications=[simulation_application],
                                                vpcConfig=vpcConfig
                                              )
    responses.append(response)

print("Created the following jobs:")
job_arns = [response["arn"] for response in responses]
for response in responses:
    print("Job ARN", response["arn"])

Created the following jobs:
Job ARN arn:aws:robomaker:us-west-2:738575810317:simulation-job/sim-02nk1r56hg67
```

- To watch the simulations in AWS RoboMaker, run the code cell under **Visualizing the simulations in RoboMaker** and then choose the **Simulation 1** link from the output.

```
In [67]: display(Markdown(generate_robomaker_links(job_arns, aws_region)))
```

Click on the following links for visualization of simulation jobs on RoboMaker Console

- [Simulation 1](#)

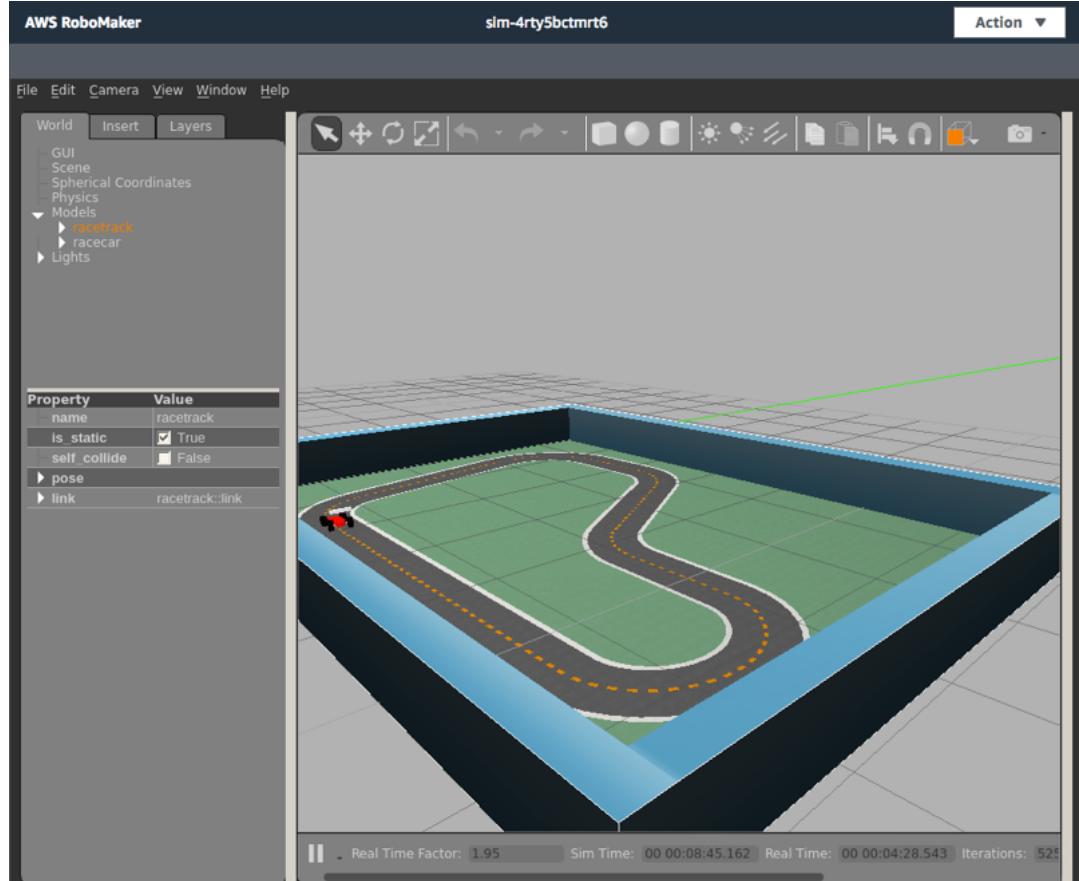
Alternatively, you can go to the AWS RoboMaker console directly to open the simulation job.

After the simulation job is initialized, the AWS RoboMaker console makes available the following visualization utilities:

- Gazebo:** an emulation of 3D worlds for simulating autonomous vehicle in the chosen track.
- rqt:** Qt-based framework and plugins for ROS GUI development.

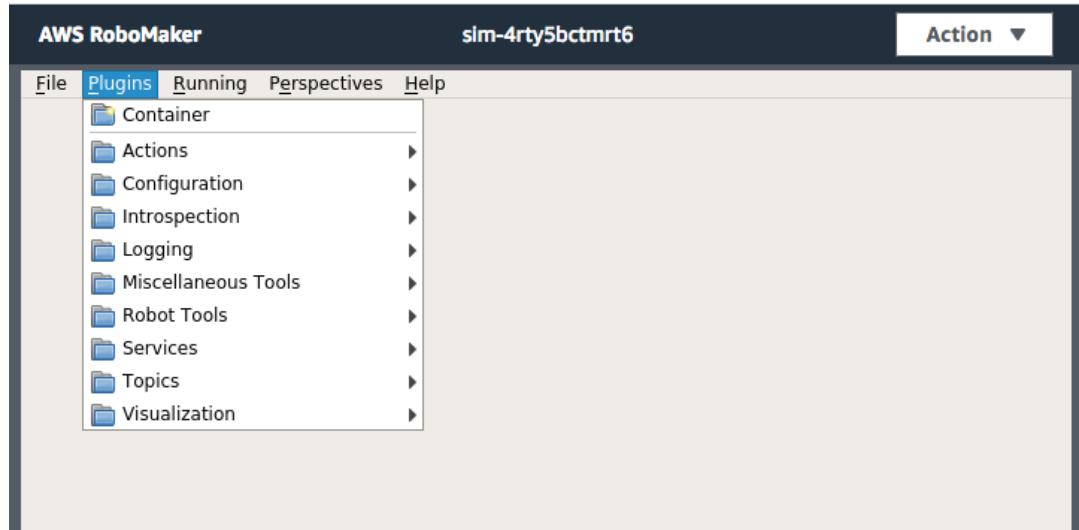
- **ivis:** ROS visualizer for displaying the field of vision as captured by the vehicle's front-facing camera.
- **Terminal:** A terminal application to provide command line access on the simulation job host.

- To view your vehicle learning in the 3D simulation, double-click or tap **Gazebo**.



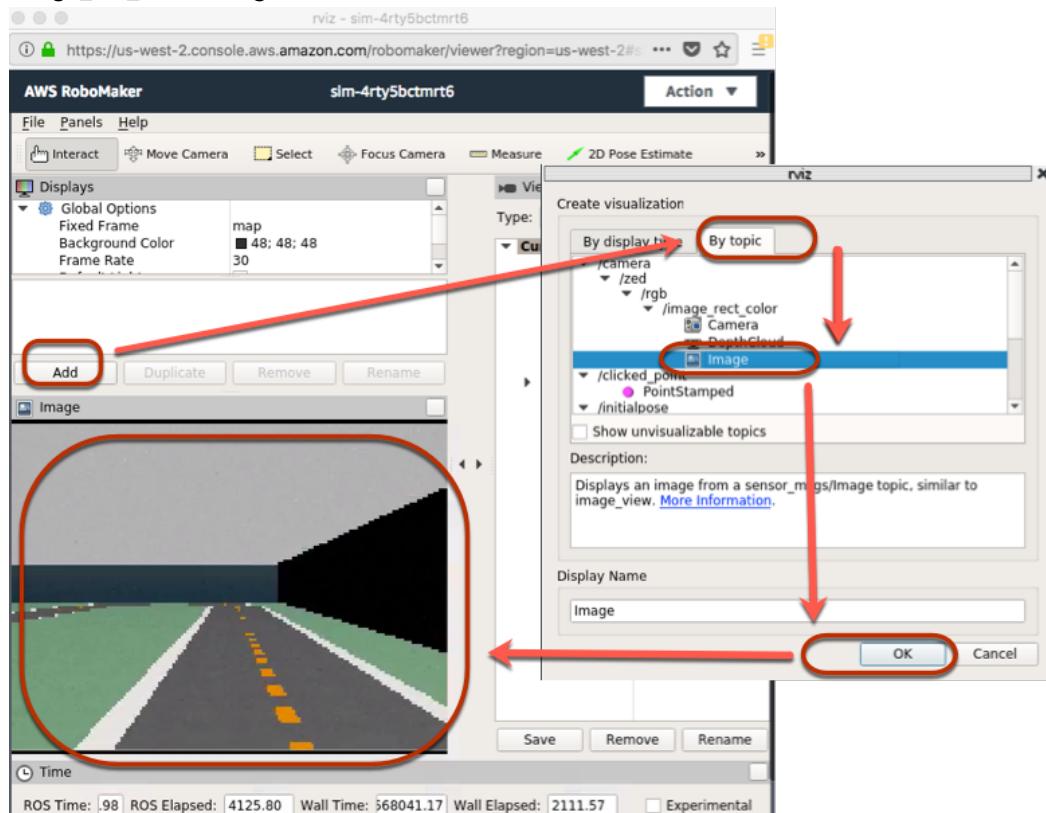
You watch the simulated vehicle navigate along the track in repeated trials starting from the starting point and ending at going off-track or reaching the finishing line. In the beginning, the vehicle can stay on the track briefly. As time goes on, it learns to stay on the track longer.

- To access **rqt** utilities, double-click or tap **rqt** and choose a plugin.

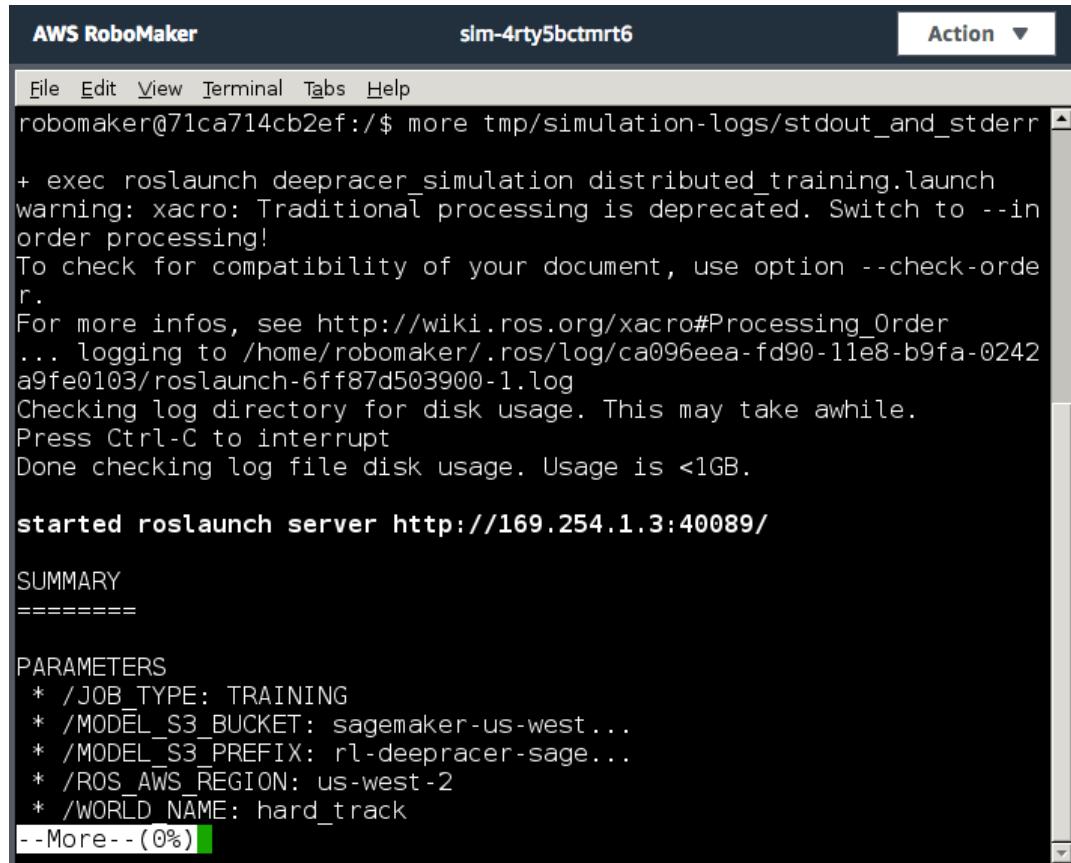


For more information about the plugins, see AWS RoboMaker plugins.

- c. To view the front-facing vision of the vehicle, double-click or tap **rviz**. Choose **Add** to create a visualization. And then choose the **By topic** tab, scroll down to choose **/camera/zed/rgb/image_rect_color/Image**, choose **OK**.



- d. To use the terminal, double-click or tap **Terminal** to open a terminal window on the simulation job host and type appropriate shell command.



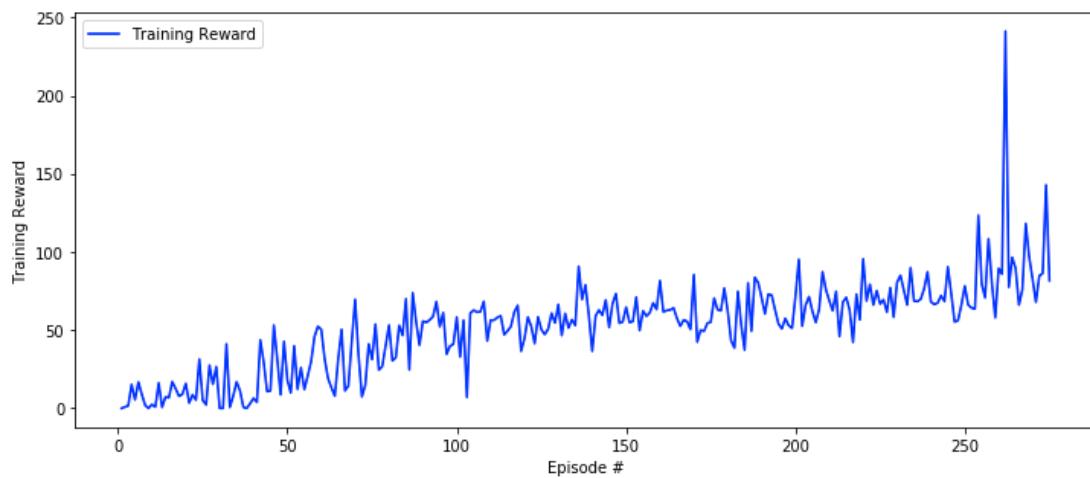
The screenshot shows a terminal window titled "AWS RoboMaker" with the session ID "slm-4rty5bctmrt6". The window has a "File Edit View Terminal Tabs Help" menu bar and an "Action" dropdown. The terminal content displays a series of Linux shell commands and their output. The output includes a warning about xacro processing being deprecated, instructions for checking compatibility, and details about the simulation setup. It shows the server started at "http://169.254.1.3:40089/" and lists parameters such as JOB_TYPE, MODEL_S3_BUCKET, MODEL_S3_PREFIX, ROS_AWS_REGION, and WORLD_NAME. A green bar at the bottom indicates "-More--(0%)".

With the simulation job host terminal opened, you can call Linux shell commands to view ([more](#) or [tail](#)) the logs or performing other operations.

To view the reward of the last 10 steps in the simulation logs, you can type the following shell command in the terminal:

```
tail /tmp/simulation-logs/stdout_and_stderr
```

6. To visualize the training performance, run the two code cells under **Plot metrics for training job**. When all is done successfully, you see a plot of **Training reward vs Episode #** similar to the following.



In this particular example, the training reward appears to start to plateau. Perhaps more data are needed to verify if it's true. If the training job is running, you can run the code cell under **Plot metrics for training job** again to include more recent data into the plot. If they persist, the onset of large fluctuations can indicate certain deficiency in the reward function. Thus, you might update the reward function definition. In any case, you need to collect more data with more training.

After training has elapsed the specified amount of time, you can locate the trained model artifacts in the training job's S3 bucket, e.g., `s3://<bucket>/<sagemaker-training-job-name>/output/model.tar.gz`. Download the model artifacts file, copy it to a USB drive and then transfer the file to your AWS DeepRacer vehicle's compute module.

- To clean up when you're done with training and no longer need the AWS RoboMaker and Amazon SageMaker resources, run the two code cells under **Clean Up**.
- To evaluate the model that has been trained thus far, run the code cell under **Evaluation**.

If successful, a simulation job is created for the task in AWS RoboMaker. Make note of the job name in the output below the code cell. You may need it to open the simulation job in the AWS RoboMaker console. This simulation job is similar to the simulation job for training. It provides the same utilities for you view the evaluation in progress in the AWS RoboMaker console. In particular, you can watch the evaluation trials in **Gazebo**.

- When you're done with evaluating the model and want to terminate the simulation application, run the code cell under **Clean Up Simulation Application Resource**.

Input Parameters of the AWS DeepRacer Reward Function

The AWS DeepRacer reward function takes a dictionary object as the input.

```
def reward_function(params) :
    reward = ...
    return float(reward)
```

The `params` dictionary object contains the following key-value pairs:

```
{
```

```
    "all_wheels_on_track (p. 58)": Boolean,      # flag to indicate if the vehicle is on the
track
    "x (p. 68)": float,                         # vehicle's x-coordinate in meters
    "y (p. 68)": float,                         # vehicle's y-coordinate in meters
    "distance_from_center (p. 61)": float,       # distance in meters from the track center
    "is_left_of_center (p. 63)": Boolean,        # Flag to indicate if the vehicle is on the
left side to the track center or not.
    "heading (p. 63)": float,                    # vehicle's yaw in degrees
    "progress (p. 63)": float,                   # percentage of track completed
    "steps (p. 66)": int,                        # number steps completed
    "speed (p. 64)": float,                      # vehicle's speed in meters per second (m/
s)
    "steering_angle (p. 64)": float,             # vehicle's steering angle in degrees
    "track_width (p. 66)": float,                 # width of the track
    "waypoints (p. 68)": [[float, float], ... ], # list of [x,y] as milestones along the
track center
    "closest_waypoints (p. 60)": [int, int]      # indices of the two nearest waypoints.
}
```

The more detailed technical reference for the input parameters is as follows.

all_wheels_on_track

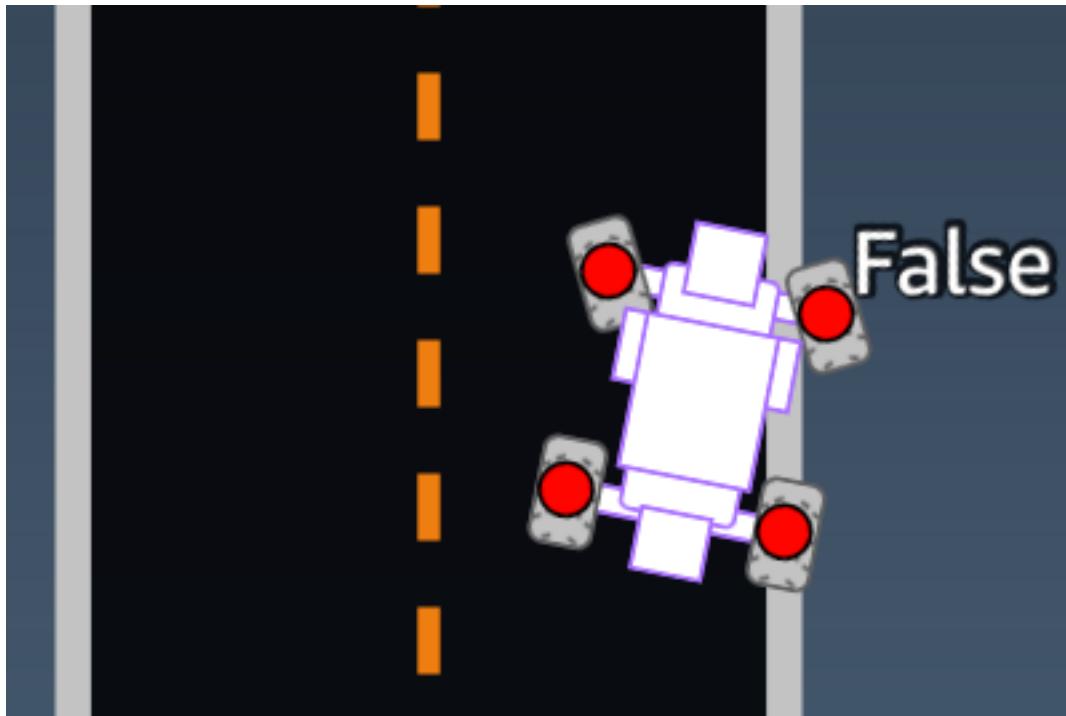
Type: Boolean

Range: True | False

A Boolean flag to indicate whether the vehicle is on-track or off-track. It's off-track (False) if any of its wheels are outside of the track borders. It's on-track (True) if all of the wheels are inside the two track borders. The following illustration shows that the vehicle is on-track.



The following illustration shows that the vehicle is off-track.



Example: A reward function using the `all_wheels_on_track` parameter

```
define reward_function(params):
#####
''' Example of using all_wheels_on_track and speed '''
'''

# Read input variables
all_wheels_on_track = params['all_wheels_on_track']
speed = params['speed']

# Set the speed threshold based your action space
SPEED_THRESHOLD = 1.0

if not all_wheels_on_track:
    # Penalize if the car goes off track
    reward = 1e-3
elif speed < SPEED_THRESHOLD:
    # Penalize if the car goes too slow
    reward = 0.5
else:
    # High reward if the car stays on track and goes fast
    reward = 1.0

return reward
```

closest_waypoints

Type: [int, int]

Range: [0:Max-2,1:Max-1]

The zero-based indices of the two neighboring waypoints closest to the vehicle's current position of (x , y). The distance is measured by the Euclidean distance from the center of the vehicle. Max is the length of the waypoints list.

Example: A reward function using the closest_waypoints parameter.

The following example reward function demonstrates how to use waypoints and closest_waypoints as well as heading to calculate immediate rewards.

```
def reward_function(params):

#####
# Example of using waypoints and heading to make the car in the right direction
# Import math
# Read input variables
waypoints = params['waypoints']
closest_waypoints = params['closest_waypoints']
heading = params['heading']

# Initialize the reward with typical value
reward = 1.0

# Calculate the direction of the center line based on the closest waypoints
next_point = waypoints[closest_waypoints[1]]
prev_point = waypoints[closest_waypoints[0]]

# Calculate the direction in radius, arctan2(dy, dx), the result is (-pi, pi) in radians
track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] - prev_point[0])
# Convert to degree
track_direction = math.degrees(track_direction)

# Calculate the difference between the track direction and the heading direction of the car
direction_diff = abs(track_direction - heading)
if direction_diff > 180:
    direction_diff = 360 - direction_diff

# Penalize the reward if the difference is too large
DIRECTION_THRESHOLD = 10.0
if direction_diff > DIRECTION_THRESHOLD:
    reward *= 0.5

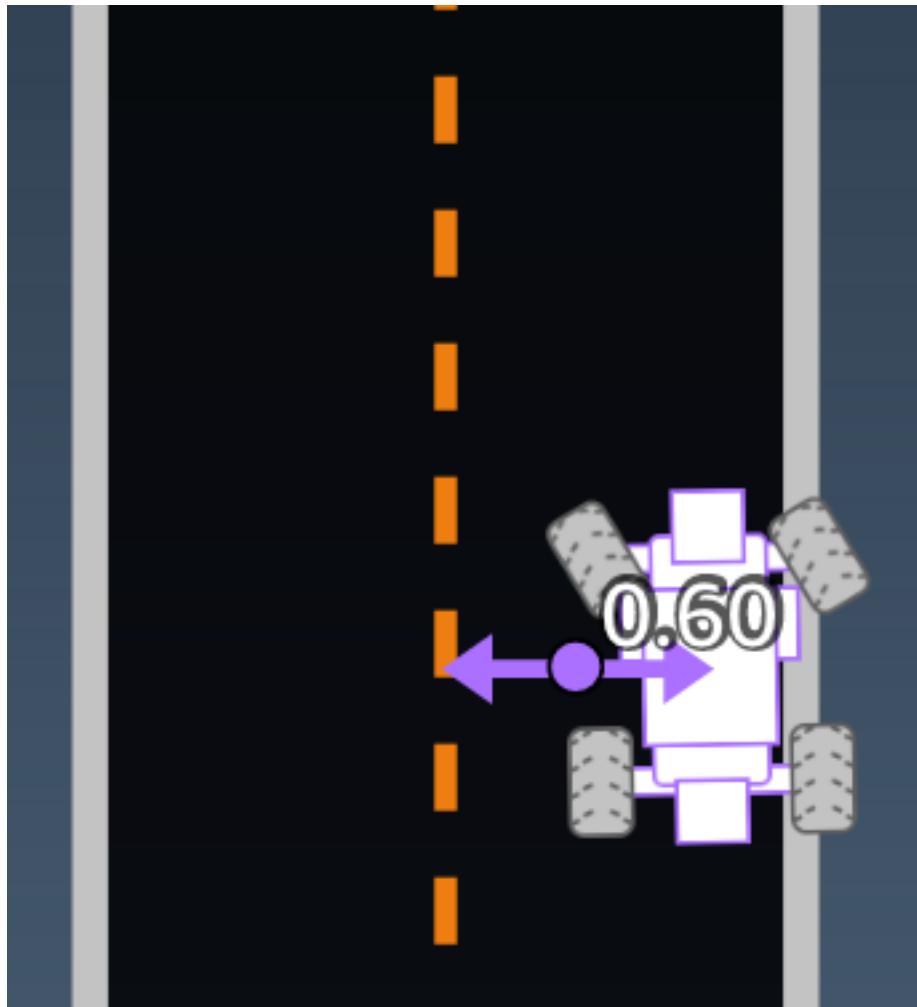
return reward
```

distance_from_center

Type: float

Range: 0 : ~track_width/2

Displacement, in meters, between the vehicle center and the track center. The observable maximum displacement occurs when any of the agent's wheels are outside a track border and, depending on the width of the track border, can be slightly smaller or larger than half the track_width.



Example: A reward function using the `distance_from_center` parameter

```
def reward_function(params):
    """
    Example of using distance from the center
    """

    # Read input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Penalize if the car is too far away from the center
    marker_1 = 0.1 * track_width
    marker_2 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    else:
        reward = 1e-3 # likely crashed/ close to off track

    return reward
```

heading

Type: float

Range: -180 : +180

Heading direction, in degrees, of the vehicle with respect to the x-axis of the coordinate system.



Example: A reward function using the heading parameter

For more information, see [closest_waypoints \(p. 60\)](#).

is_left_of_center

Type: Boolean

Range: True | False

A Boolean flag to indicate if the vehicle is on the left side to the track center (True) or on the right side (False).

progress

Type: float

Range: 0 : 100

Percentage of track completed.

Example: A reward function using the progress parameter

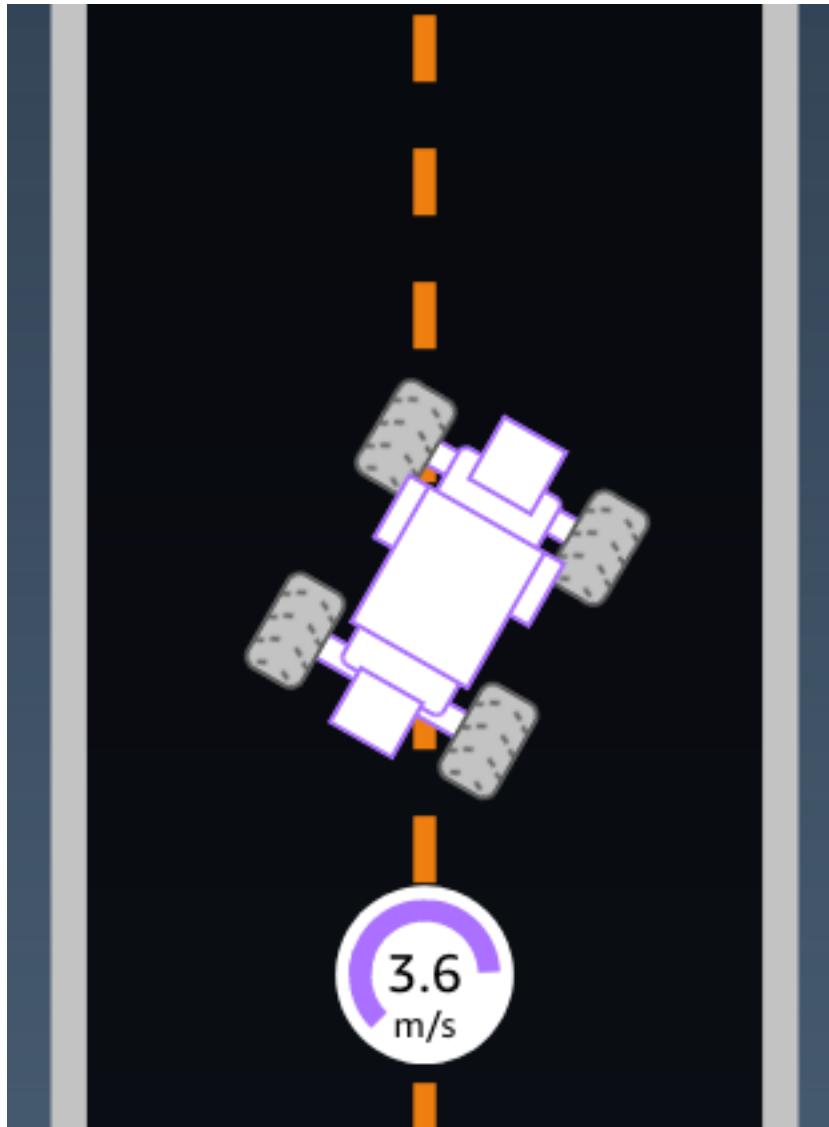
For more information, see [steps \(p. 66\)](#).

speed

Type: float

Range: 0.0:5.0

The observed speed of the vehicle, in meters per second (m/s).



Example: A reward function using the speed parameter

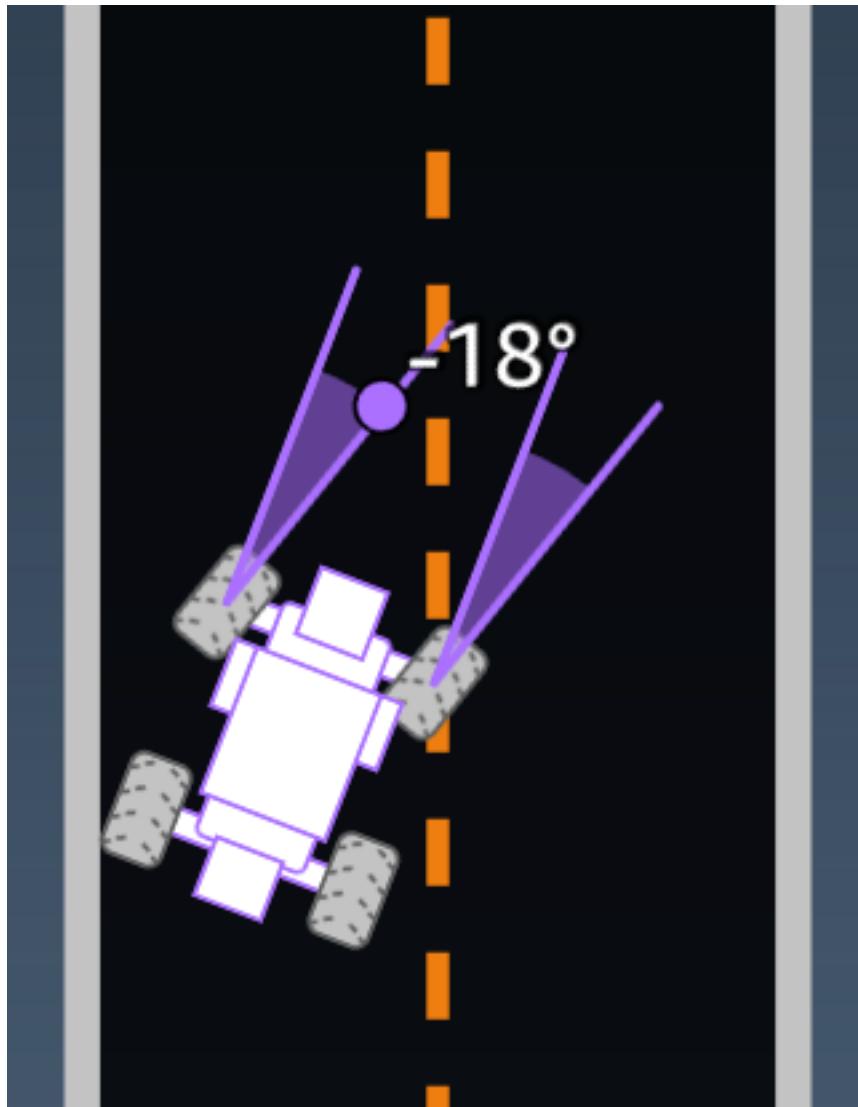
For more information, see [all_wheels_on_track \(p. 58\)](#).

steering_angle

Type: float

Range: -30:30

Steering angle, in degrees, of the front wheels from the center line of the vehicle. The negative sign (-) means steering to the right and the positive (+) sign means steering to the left. The vehicle center line is not necessarily parallel with the track center line as is shown in the following illustration.



Example: A reward function using the `steering_angle` parameter

```
def reward_function(params):
    ...
    Example of using steering angle
    ...

    # Read input variable
    steering = abs(params['steering_angle']) # We don't care whether it is left or right
    steering

    # Initialize the reward with typical value
    reward = 1.0

    # Penalize if car steer too much to prevent zigzag
```

```
STEERING_THRESHOLD = 20.0
if steering > ABS_STEERING_THRESHOLD:
    reward *= 0.8

return reward
```

steps

Type: int

Range: 0 : N_{step}

Number of steps completed. A step corresponds to an action taken by the vehicle following the current policy.

Example: *A reward function using the steps parameter*

```
def reward_function(params):
    """
    Example of using steps and progress
    """

    # Read input variable
    steps = params['steps']
    progress = params['progress']

    # Total num of steps we want the car to finish the lap, it will vary depends on the track
    # length
    TOTAL_NUM_STEPS = 300

    # Initialize the reward with typical value
    reward = 1.0

    # Give additional reward if the car pass every 100 steps faster than expected
    if (steps % 100) == 0 and progress > (steps / TOTAL_NUM_STEPS) * 100 :
        reward += 10.0

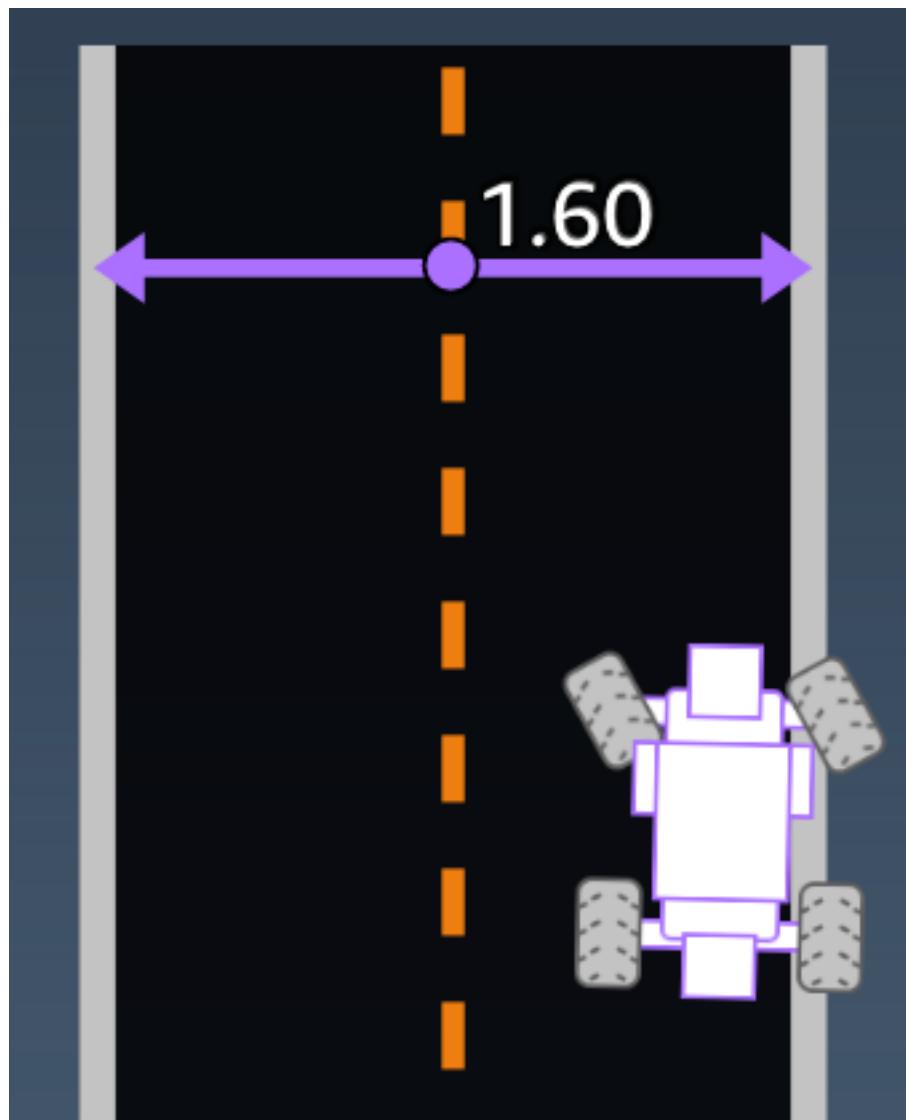
    return reward
```

track_width

Type: float

Range: 0 : D_{track}

Track width in meters.



Example: A reward function using the `track_width` parameter

```
def reward_function(params):
    """
    Example of using track width
    """

    # Read input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Calculate the distance from each border
    distance_from_border = 0.5 * track_width - distance_from_center

    # Reward higher if the car stays inside the track borders
    if distance_from_border >= 0.05:
        reward *= 1.0
    else:
        reward = 1e-3 # Low reward if too close to the border or goes off the track
```

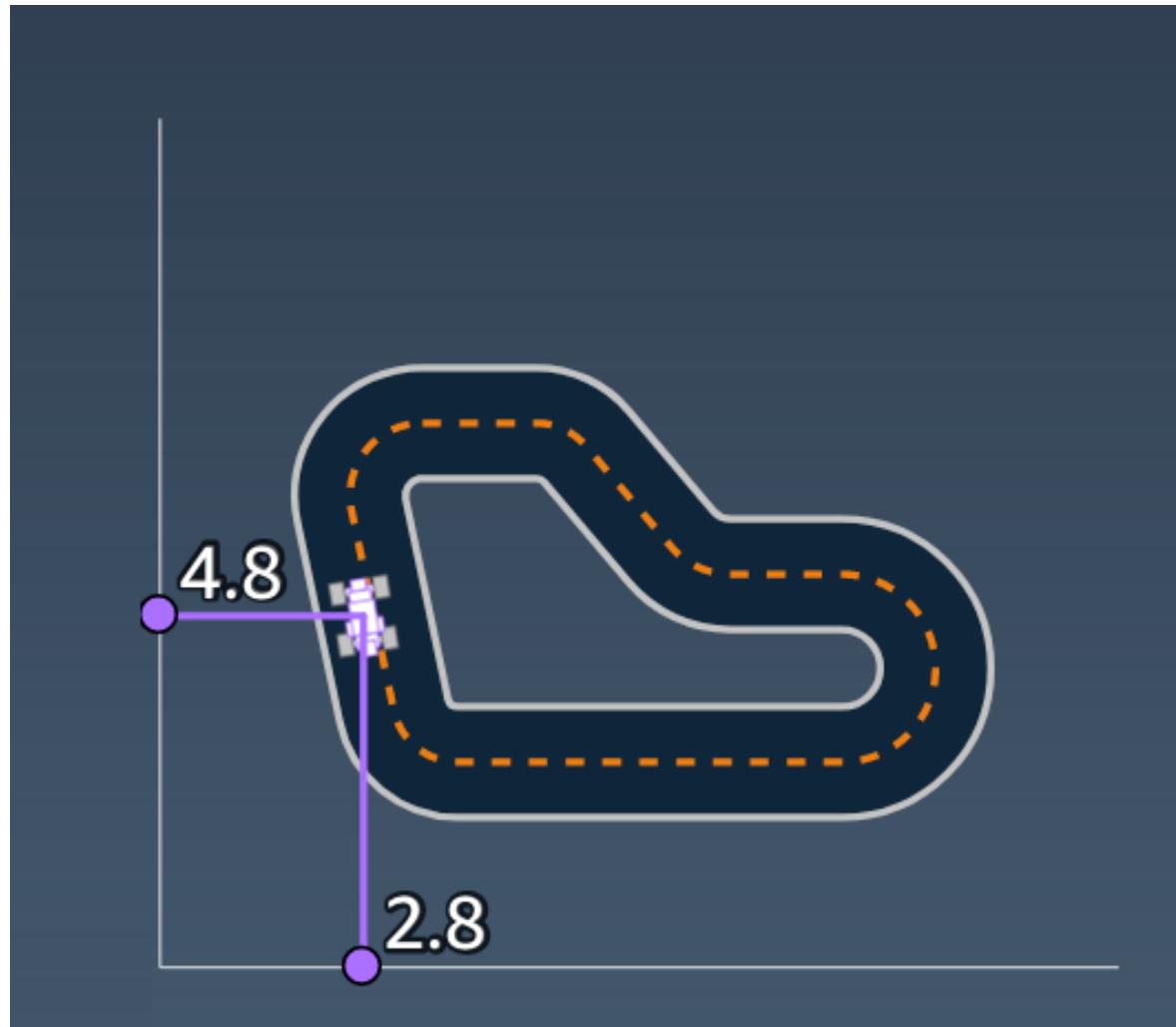
```
return reward
```

x, y

Type: float

Range: 0 : N

Location, in meters, of the vehicle center along the x and y axes, of the simulated environment containing the track. The origin is at the lower-left corner of the simulated environment.

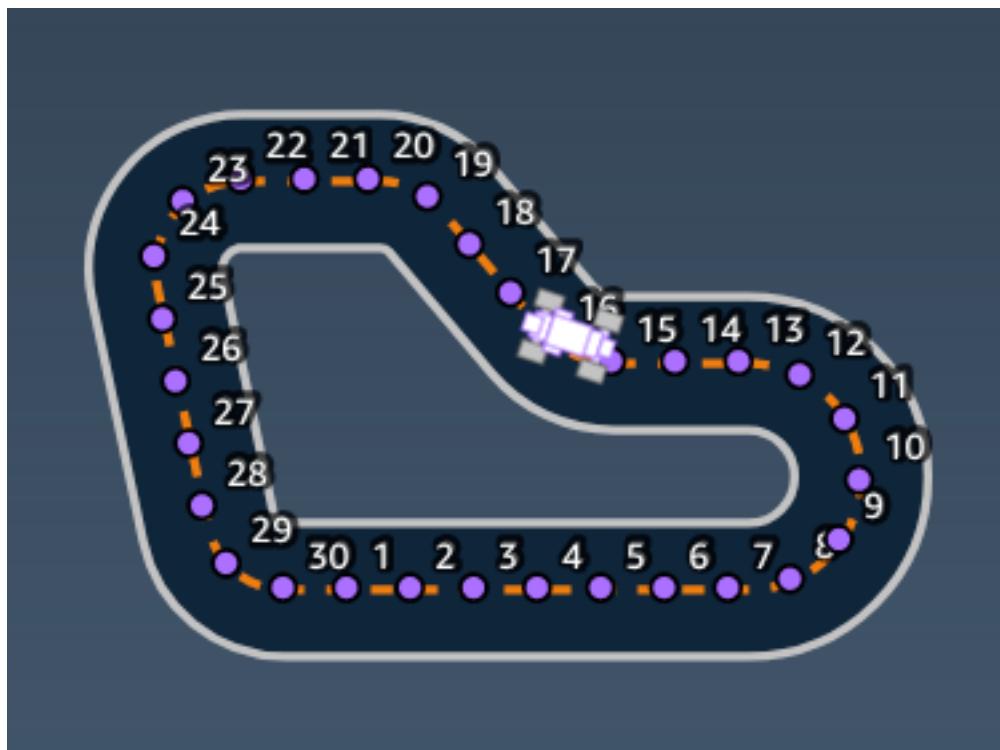


waypoints

Type: list of [float, float]

Range: [[$x_{w,0}, y_{w,0}$] ... [$x_{w,Max-1}, y_{w,Max-1}$]]

An ordered list of track-dependent Max milestones along the track center. Each milestone is described by a coordinate of ($x_{w,i}, y_{w,i}$). For a looped track, the first and last waypoints are the same. For a straight or other non-looped track, the first and last waypoints are different.



Example A reward function using the waypoints parameter

For more information, see [closest_waypoints \(p. 60\)](#).

Operate Your AWS DeepRacer Vehicle

After you finish training and evaluating an AWS DeepRacer model in the AWS DeepRacer simulator, you can deploy the model to your AWS DeepRacer vehicle. You can set the vehicle to drive on a track and evaluate the model's performance in a physical environment. This mimics a real-world autonomous race.

Before driving your vehicle for the first time, you must set up the vehicle, install software updates, and calibrate its drive-chain sub-system.

To drive your vehicle on a physical track, you must have a track. For more information, see [Build Your Physical Track \(p. 100\)](#)

Topics

- [Prep Your AWS DeepRacer Vehicle \(p. 70\)](#)
- [Choose a Wi-Fi Network for Your AWS DeepRacer Vehicle \(p. 81\)](#)
- [Calibrate Your AWS DeepRacer Vehicle \(p. 85\)](#)
- [Upload a Model to Your AWS DeepRacer Vehicle \(p. 91\)](#)
- [Drive Your AWS DeepRacer Vehicle \(p. 92\)](#)
- [Manage Your AWS DeepRacer Vehicle Settings \(p. 94\)](#)
- [View Your AWS DeepRacer Vehicle Logs \(p. 98\)](#)

Prep Your AWS DeepRacer Vehicle

Your AWS DeepRacer vehicle is a Wi-Fi enabled, battery-powered, 1/18th-scale model four-wheel drive car with a front-mounted 4-megapixel camera and an Ubuntu-based compute module.

The vehicle can drive autonomously by running inference that is based on a reinforcement learning model in its compute module. You can also drive the vehicle manually, without deploying any reinforcement learning model. If you have not already obtained an AWS DeepRacer vehicle, you can [order one here](#).

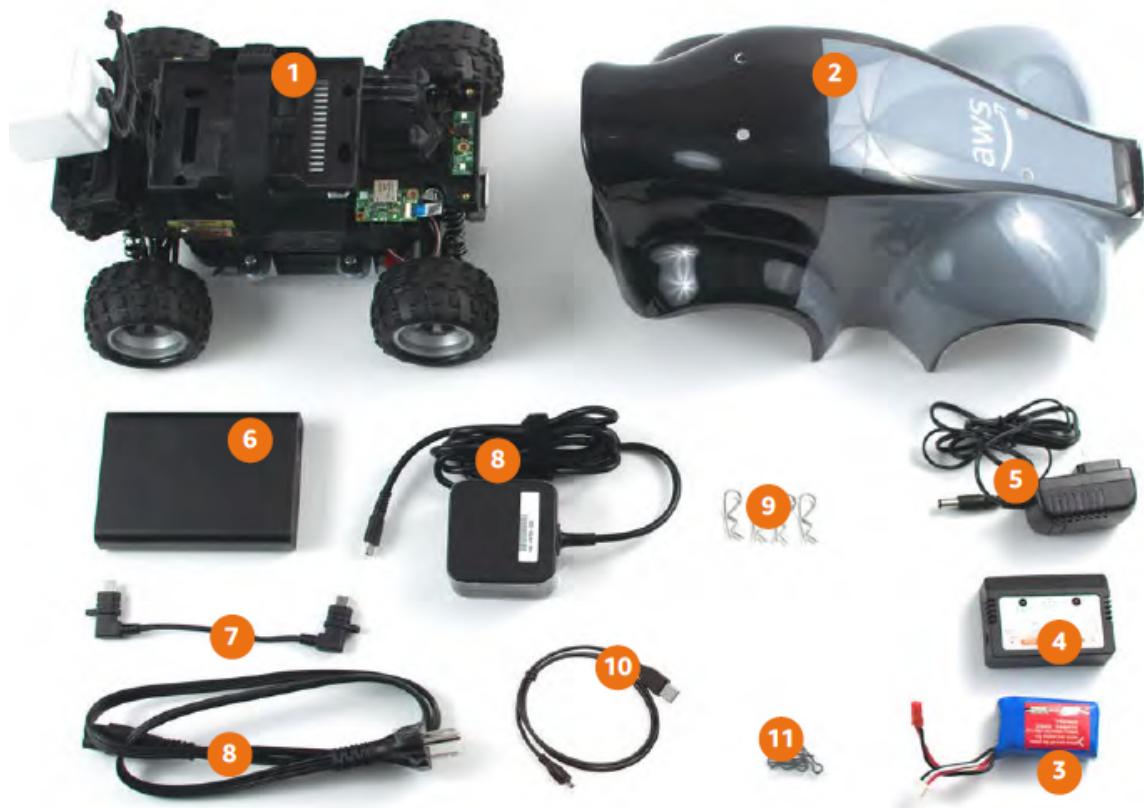
The AWS DeepRacer vehicle is powered by a brushed motor. The driving speed is controlled by a voltage regulator that controls how fast the motor spins. The [servomechanism \(servo\)](#) that operates the steering system is protected by the black cover in the AWS DeepRacer vehicle chassis.

Topics

- [Inspect Your AWS DeepRacer Vehicle \(p. 71\)](#)
- [Charge Your AWS DeepRacer Vehicle \(p. 72\)](#)
- [Test Your AWS DeepRacer Compute Module \(p. 73\)](#)
- [Turn Off Your AWS DeepRacer Vehicle \(p. 74\)](#)
- [AWS DeepLens Vehicle Chassis Parts \(p. 74\)](#)

Inspect Your AWS DeepRacer Vehicle

When you open your AWS DeepRacer vehicle box, you should find the following components and accessories:



Components	Comments
Vehicle Chassis [1]	Includes a front-mounted camera for capturing vehicle driving experiences and the compute module for autonomous driving. You can view images captured by the camera as a streaming video on the vehicle's device console. The chassis includes a brushed electric motor, an electronic speed controller (ESC), and a servomechanism (servo)
Vehicle body shell [2]	Remove this when setting up the vehicle.
Vehicle battery [3]	A 7.4v Li-Po battery pack to power the motor.
Vehicle battery charger [4]	Use this to charge the vehicle battery that powers the vehicle drive chain.
Vehicle battery power adapter [5]	Use this to connect the vehicle battery charger to a power outlet.

Components	Comments
Compute module power bank [6]	Use this to power the compute module that runs inference on a downloaded AWS DeepRacer reinforcement learning model.
Compute module power bank connector cable [7]	Use this USB C-to-USB C cable to connect the compute module with the power bank.
Power cord and adapter [8]	Use this to charge the compute module power bank and the compute module.
Vehicle chassis pins [4] and [11]	Use the four white pins and the spare ones (in black) to fasten the compute module to the vehicle chassis.
USB-to-μUSB cable [10]	Use this to support USB-OTG functionality.

To set up your AWS DeepRacer vehicle, you must also have the following items ready:

- A computer with a USB port and access to the internet.
- A Wi-Fi network connected to the internet.
- An AWS account.

Now follow the instructions in the [next section \(p. 72\)](#) to make sure your vehicle battery and the power bank are charged.

Charge Your AWS DeepRacer Vehicle

Your AWS DeepRacer vehicle has two power sources. The vehicle battery powers the motor to move the vehicle. The power bank keeps the compute module running. The module maintains the Wi-Fi connection, runs inference against a deployed AWS DeepRacer model, and issues a command for the vehicle to take an action.

The vehicle battery pack comes with a blue cover and has two cables. The red cable connects to the ESC and the blue cable connects to the charger. For driving, only the red cable is connected to the vehicle. After fully charged, the battery voltage will drop as the batteries discharge. When the voltage drops, the available torque also drops. This means the same speed setting results in slower actual speed. When the battery is fully empty, the vehicle stops moving. For normal autonomous driving, the battery usually lasts 15-25 minutes. To ensure consistent behavior, charge the battery after every 15 minutes of use.

To install and charge the vehicle battery and the power bank, follow the steps below.

1. Remove your AWS DeepRacer vehicle shell.
2. Remove the four vehicle chassis pins. Carefully lift vehicle chassis while keeping wires connected.
3. To charge the vehicle battery, do the following:
 - a. To connect the vehicle battery to your vehicle, plug the red end of the vehicle battery cable into the black and red cable connector on your vehicle.
 - b. To secure the vehicle battery, tie the battery under the vehicle chassis with the attached straps.

Make sure to keep all the cables inside the vehicle.

 - c. To check if the vehicle battery is charged, do the following:
 - i. Slide the vehicle power switch to turn on the vehicle.

- ii. Listen for two short beeps.

If you don't hear the beeps, the vehicle is not charged. To charge the vehicle battery, see [the section called "How to Charge the AWS DeepRacer Drive Module Battery?" \(p. 117\)](#).

- iii. Slide the vehicle power switch back to turn off the vehicle.

4. To check the power bank charging level, do the following:

- a. Press the power button on the power bank.
- b. Check the four LED lights next to the power button to determine the charging level.

If all the four LED lights are lit, the power bank is fully charged. If none of the LED lights are lit, the power bank needs to be charged.

- c. To charge the power bank, insert the USB C plug from the power adapter into the USB C port of the power bank. It takes some time for the power bank to be fully charged. When it is charged, repeat **Step 4** to confirm that the power bank is fully charged.

5. To install the power bank, do the following:

- a. Insert the power bank into its holder with the power button and USB C port facing the back of the vehicle.
- b. Use the strap to tie the power bank to the vehicle chassis securely.

Note

Do not connect the power bank to the compute module in this step.

Test Your AWS DeepRacer Compute Module

Test the compute module to verify that it can be started successfully. To test the module by using an external power source, follow the steps below:

To test your vehicle's compute module

1. Connect the compute module to a power source. Connect the power cord to the power adapter, plug the power cord to a power outlet, and insert the power adapter's USB C plug into the USB C port on the compute module.
2. Turn on the vehicle's compute module by pressing the power button on the compute module.
3. To verify the compute module's status, check that the LED lights are shown as follows:

- Solid blue

The compute module is started, connected to the specified Wi-Fi, and ready to go.

In this state, you can log in to the compute module after you attach it to a monitor using an HDMI cable, a USB mouse and a USB keyboard. For the first-time login, use `deepracer` for both the **username** and **password**. You will then be asked to reset the password for future logins. For security reasons, choose a strong password phrase for the new password.

- Blinking red

The compute module is in setup mode.

- Solid yellow

The compute module is initializing.

- Solid red

The compute module failed to connect to the Wi-Fi network.

- When you're done with the test, press the power button on the compute module to turn it off and then unplug it from the external power source.

Turn Off Your AWS DeepRacer Vehicle

To turn off your AWS DeepRacer vehicle, unplug the vehicle from the external power source. You can also press the power button on the device until power indicator is off.

AWS DeepLens Vehicle Chassis Parts

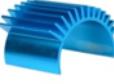
The AWS DeepLens vehicle uses [WLToys A949](#) and [A979](#) Remote Control (RC) car chassis. To help you find replacement when any part of the chassis is damaged, we provide below a catalogue of the parts.

AWS DeepLens Vehicle Chassis Parts

Part	Name
	Tire
	Front bumper
	Suspension arm
	Pull rod
	C style seat

Part	Name
	Transmission shaft
	Round head screw, M2x17.5mm
	Chassis car bottom
	Turning seat
	Rear suspension frame
	Metal hexagonal combiner set
	Gear box shell

Part	Name
	Differential box case
	Differential drive cup
	Front rear shelter
	Servo seat
	Central driving shaft
	Shock frame
	Servo arm

Part	Name
	Differential mechanism
	Reduction gear
	Motor base
	Lithium battery 7.4V 1100mah
	17g steering engine
	Motor heat dissipation cover
	Stub axle

Part	Name
	Screw gasket of motor, fixed seat
	390 motor
	Hexagon connector 4x8x3mm
	Hexagon connector 8x12x3.5mm
	Ball bearing 7x11x3mm
	Ball bearing 8x12x3.5mm
	Middle axle disc plate

Part	Name
	Screw 2.6x6mm
	Screw 2x7mm
	Screw 2.5x8mm
	Screw 2x16mm
	Screw 2.5x6x6mm
	Screw M3x5mm
	Ball screw 10.8x4mm

Part	Name
	Screw 2x6mm
	Screw 2x9.5mm
	M3 locknut
	Axle hinge pin
	Drive shaft
	Swing arm pin
	Screw 2*29KM
	Hair pin

Part	Name
	Front shock absorber
	Charger
	Metal gear
	Back shock absorber
	ESC

Choose a Wi-Fi Network for Your AWS DeepRacer Vehicle

The first time you open your AWS DeepRacer vehicle, you must set it up to connect to a Wi-Fi network. Complete this setup to get the vehicle's software updated and to get the IP address to access the vehicle's device console.

This section walks you through the steps to perform the following tasks:

- Connect your laptop or desktop computer to your vehicle.
- Set up the vehicle's Wi-Fi connection.
- Update the vehicle's software.
- Get the vehicle's IP address.
- Test drive the vehicle.

Use a laptop or desktop computer to perform the setup tasks. We'll refer to this setup computer as your computer, to avoid possible confusion with the vehicle's compute module, which is running the Ubuntu operating system.

After the initial setup of the Wi-Fi connection, you can follow the same instructions to choose a different Wi-Fi network.

Topics

- [Get Ready to Choose a Wi-Fi Network for Your AWS DeepRacer Vehicle \(p. 82\)](#)
- [Set Up Wi-Fi Connection and Update Your AWS DeepRacer Vehicle's Software \(p. 82\)](#)
- [Sign In to AWS DeepRacer Vehicle's Device Console \(p. 83\)](#)

Get Ready to Choose a Wi-Fi Network for Your AWS DeepRacer Vehicle

To set up your vehicle's Wi-Fi connection, connect your a laptop or desktop computer to your vehicle's compute module using the included *USB-to-USB C* cable.

To connect your computer to your vehicle's compute module, follow the steps below.

To connect your computer to your vehicle to set up the device

1. Insert the USB end of the *USB-to-USB C* cable into your computer's USB port.
2. Insert the cable's USB C end into your vehicle's USB C port.

You're now ready to proceed to setting up your vehicle's Wi-Fi connection.

Set Up Wi-Fi Connection and Update Your AWS DeepRacer Vehicle's Software

Before you follow the steps here to set up the Wi-Fi connection, be sure you complete the steps in [the section called "Get Ready for Choose a Wi-Fi" \(p. 82\)](#).

1. Look at the bottom of your vehicle and make note of the password printed under **Host name**. You'll need it to log in to the device control console to perform the setup.
2. On your computer, go to <https://deepracer.aws> to launch the device control console of your vehicle.
3. When prompted with a message that the connection is not private or secure, do one of the following.
 - a. In Chrome, choose **Advanced** and then choose **Proceed to <device_console_ip_address> (unsafe)**.
 - b. In Safari, choose **Details**, follow the **visit this website** link, and then choose **Visit Websites**. If prompted for your password to update the certificate trust settings, type the password and then choose **Update settings**.
 - c. In Opera, choose **Continue Anyway** when warned of an invalid certificate.
 - d. In Edge, choose **Details** and then choose **Go on to the webpage (Note recommended)**.
 - e. In Firefox, choose **Advanced**, choose **Add Exception**, and then choose **Confirm Security Exception**.
4. Under **Unlock your AWS DeepRacer vehicle**, enter the password noted in **Step 1** and then choose **Access vehicle**.

5. On the **Connect your vehicle to your Wi-Fi network** pane, choose your Wi-Fi network name from the **Wi-Fi network name (SSID)** drop-down menu, type the password of your Wi-Fi network under **Wi-Fi password**, and choose **Connect**.
6. Wait until the Wi-Fi connection status changes from **Connecting to Wi-Fi network...** to **Connected**. Then, choose **Next**.
7. On the **Software update** pane, if a software update is required, turn on the vehicle's compute module, with the included power cord and power adapter, and then choose **Install software update**.

Powering the vehicle with an external power source helps avoid interruption of the software update if the compute module's power bank become discharged.
8. Wait until the software update status changes from **Installing software update** to **Software update installed successfully**.
9. Note the IP address shown under **Wi-Fi network details**. You'll need it to open the vehicle's device control console after the initial setup and any subsequent modification of the Wi-Fi network settings.

Sign In to AWS DeepRacer Vehicle's Device Console

After you set up the vehicle's Wi-Fi connection and install required software updates, you should access the vehicle to verify if the vehicle's Wi-Fi connection is working. The process involves logging in to your vehicle's device console with the Wi-Fi connection to the vehicle's IP address.

The device control console is hosted on the vehicle and is accessed with the IP address you obtained at the end of the [Wi-Fi setup \(p. 82\)](#) section.

To access the device console of your AWS DeepRacer vehicle through the Wi-Fi connection

1. To access the device console of your vehicle, open a web browser on your computer, tablet or a smart phone and type your vehicle's IP address into the address bar.

You can get this IP address when [setting up the vehicle's Wi-Fi connection \(p. 82\)](#). For illustration, we use 10.92.206.61 as an example.

If you are prompted with a warning that the connection is not private or secure, ignore the message and continue to connect to the device console.

2. Under **Unlock your AWS DeepRacer vehicle**, type the device console's password in **Password** and then choose **Access vehicle**.



The default AWS DeepRacer password can be found printed on the bottom of your vehicle.

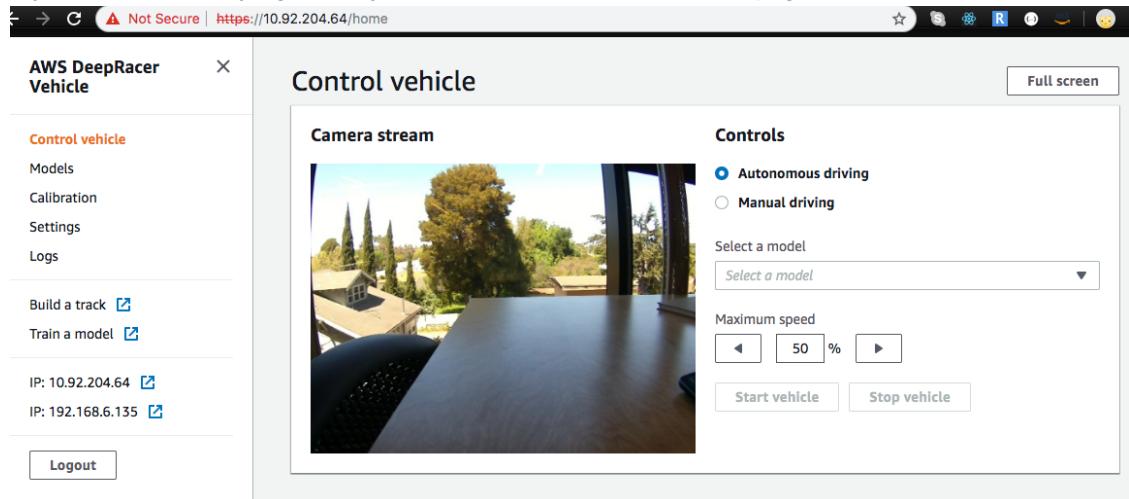
Password

Access vehicle

[Forgot password](#)

You can find the default password printed on the bottom of your vehicle (under **Host Name**).

3. If you are successfully signed in, you see the device console's home page as follows.



AWS DeepRacer Vehicle

Control vehicle

Camera stream

Controls

Autonomous driving (radio button selected)

Manual driving (radio button)

Select a model

Maximum speed

Start vehicle | Stop vehicle

You're now ready to set up and operate your vehicle. If this is your first time operating the vehicle, proceed to [calibrating the vehicle \(p. 85\)](#) now.

Calibrate Your AWS DeepRacer Vehicle

To achieve the best performance, it's essential that you calibrate some physical parts of your AWS DeepRacer vehicle. If you use an uncalibrated vehicle, it can add uncertainty when testing your model. If the vehicle's performance is not optimal, you might be tempted to only adjust the deep learning model code. However, you won't be able to improve the vehicle performance if the root cause is mechanical. Adjust the mechanics by calibration.

To calibrate your AWS DeepRacer vehicle, set the [duty cycle](#) range for the vehicle's electronic control system (ECS) and its servomechanism (servo), respectively. Both the servo and ECS accept [pulse-width modulation \(PWM\)](#) signals as control input from the vehicle's compute module. The compute module adjusts both of the vehicle's speed and steering angle by changing the duty cycles of the PWM signals.

The maximum speed and steering angle defines the span of the action space. You can specify the maximum speed and maximum steering angle during training in simulation. When deploying the trained model to your AWS DeepRacer vehicle for driving on a real-world track, the maximum speed and steering angle of the vehicle must be calibrated to match those used in the simulation training.

To ensure that the real-world experiences match the simulated experiences, you should calibrate your vehicle to match the maximum speed and maximum steering angles between the simulation and the real world. In general, there are two ways to do this calibration:

- Define the action space in training and calibrate the physical vehicle to match the settings.
- Measure the actual performance of your vehicle and change the settings of the action space in the simulation.

A robust model can handle certain differences between the simulation and the real world. However, you should experiment with either approach and iterate to find the best results.

Before starting the calibration, turn on the compute module. After it's started and the power LED has turned solid blue, turn on the vehicle battery. After you hear two short beeps and one long beep, you're ready to proceed with the calibration.

To calibrate your AWS DeepRacer vehicle to match the training settings:

1. Follow [these instructions \(p. 83\)](#) to access your vehicle and open the device control console.
2. Choose **Calibration** from the main navigation pane.

The screenshot shows the AWS DeepRacer Device Control Console interface. On the left is a sidebar with the title "AWS DeepRacer Vehicle" and a close button "X". Below the title are links: "Control vehicle", "Models", "Calibration" (which is highlighted in orange), "Settings", and "Logs". Underneath these are "Build a track" (with a link icon), "Train a model" (with a link icon), and IP addresses "IP: 10.92.204.64" and "IP: 192.168.6.135" (each with a link icon). At the bottom of the sidebar is a "Logout" button. The main content area is titled "Calibration" and contains the instruction "Calibrate your vehicle to improve its accuracy, reliability and driving behaviors. [Learn more](#)". Below this is a "Steering" section with three fields: "Center" (-2), "Maximum left steering angle" (22), and "Maximum right steering angle" (-19). There is a "Calibrate" button to the right of the section. Below the "Steering" section is a "Speed" section with three fields: "Stopped" (-3), "Maximum forward speed" (36), and "Maximum backward speed" (-42). There is a "Calibrate" button to the right of the "Speed" section.

3. On the **Calibration** page, choose **Calibrate** in **Steering** and then follow the steps below to calibrate the vehicle's maximum steering angles.

- a. Set the vehicle on the ground or another hard surface where you can see the wheels during the steering calibration. Choose **Next**.

Steering a vehicle on a track requires the much smaller steering angles than turning wheels in the air. To measure the actual steering angles of the wheels, it's important that you place the vehicle down on the track surface.

- b. Under **Center steering**, gradually move the slider or press the left or right arrow to the position where at least one of the front wheels is aligned with the rear wheel on the same side. Choose **Next**.

AWS DeepRacer uses [Ackermann front-wheel steering](#) to turn wheels on the inside and outside of a turn. This means that the left and right front wheels generally turn at different angles. In AWS DeepRacer, the calibration is done on the center value. Therefore, you need to adjust the wheels on the selected side to be aligned in a straight line.

Note

Make sure to [calibrate your AWS DeepRacer vehicle well \(p. 85\)](#) so that it can maintain center steering as straight as possible. You can test this by manually pushing the vehicle to verify it follows a straight path.

- c. Under **Maximum left steering**, gradually move the slider to the left or press the left arrow until the vehicle front wheels stops turning left. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum left steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Next**.

Calibration > Calibrate steering angle

Step 1 Set your vehicle on the ground

Step 2 Calibrate center

Step 3 Calibrate maximum left steering

Step 4 Calibrate maximum right steering

Calibrate steering angle

Maximum left steering

Increase the Value to turn the front wheels to the left until they stop turning.

Value

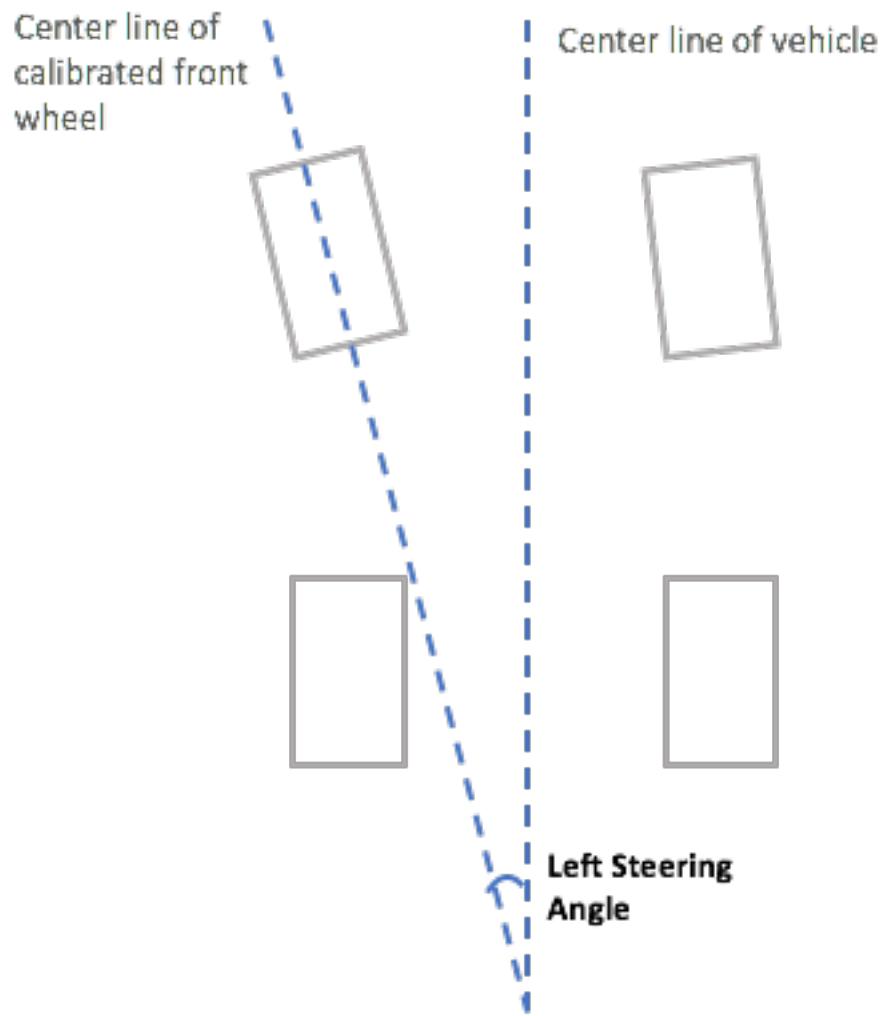
Estimated angle: 26-32°

Front

Back

Cancel Previous Next

To measure the actual maximum left steering angle, draw a center line for the vehicle, mark the two edge points of the selected front wheel for calibration, and draw the center line of this front wheel until it crosses over the center line of the vehicle. Use a protractor to measure the angle. See the figure below. If you want to match the actual angle in your training, you can set the same value in the action space in your next training job.



- d. Under **Maximum right steering**, gradually move the slider to the right until the selected front wheels stop turning right. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum right steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Done**.

Calibration > Calibrate steering angle

Step 1 Set your vehicle on the ground

Step 2 Calibrate center

Step 3 Calibrate maximum left steering

Step 4 Calibrate maximum right steering

Calibrate steering angle

Maximum right steering

Decrease the Value to turn the front wheels to the right until they stop turning.

Value

Estimated angle: 26-32°

Front

Back

Cancel Previous Done

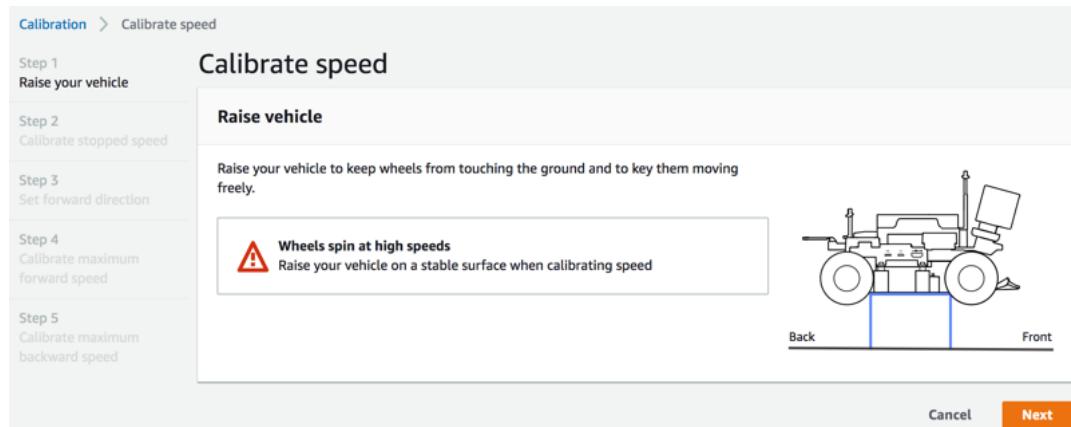
A screenshot of the AWS DeepRacer calibration interface. The current step is 'Step 4 Calibrate maximum right steering'. The main panel shows a slider for 'Value' ranging from -10 to -50, with the current setting at -18. An arrow icon points upwards next to the slider. To the right is a diagram of the vehicle with its front wheels highlighted in blue. The text 'Front' is above the front wheels and 'Back' is below the rear wheels. At the bottom are buttons for 'Cancel', 'Previous', and 'Done'.

To measure the actual maximum right steering angle, follow the steps similar to those used to measure the maximum left steering angle.

This concludes the steering calibration for your AWS DeepRacer vehicle.

4. To calibrate the vehicle's maximum speed, choose **Calibrate in Speed** on the **Calibration** page and then follow the steps below.

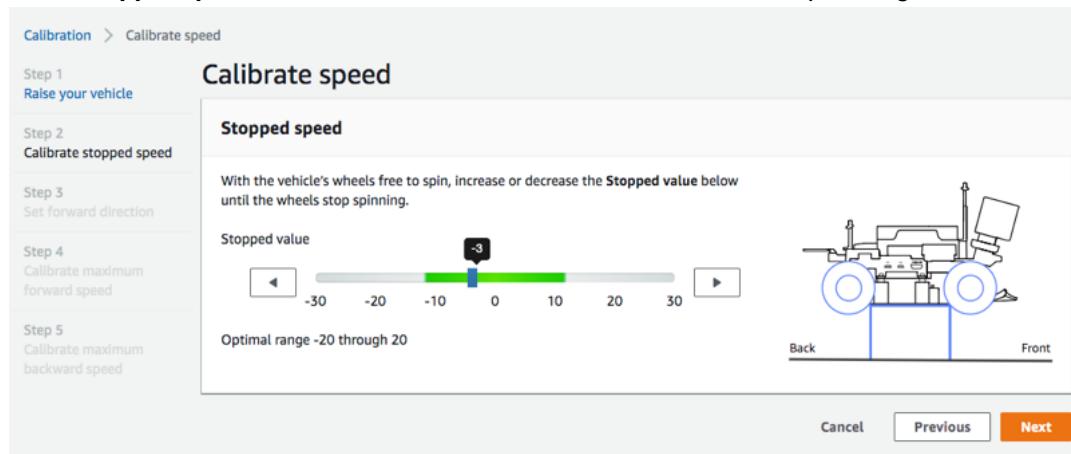
- a. Raise the vehicle so that the wheels are free to turn. Choose **Next** on the device control console.



Note

If the vehicle's speed has been set too high, it may run too fast during calibration and cause damage to the environment, the vehicle, or others nearby. You should raise the vehicle, as instructed here, but not hold it in your hands.

- b. To calibrate the stopped speed, press the left or right arrow to gradually change **Stopped value** under **Stopped speed** on the device control console until the wheels stop turning. Choose **Next**.



Note

When pressing the **Stopped value** further left or further right to the value when you start hearing noises, the wheels are about to move. The ideal zero-throttle point is the middle of the two values. For example, if you start hearing a noise at 16 on the left and at -4 on the right, the optimal stopped value should be 10.

- c. To set the vehicle's forward direction, place the vehicle as shown on the screen and the image here, and then press the left or right arrow to make the wheels turn. If the wheels turn clockwise, the forward direction is set. If not, toggle **Reverse direction**. Choose **Next**.

Calibration > Calibrate speed

Step 1 Raise your vehicle

Step 2 Calibrate stopped speed

Step 3 Set forward direction

Step 4 Calibrate maximum forward speed

Step 5 Calibrate maximum backward speed

Calibrate Speed

Set forward direction

Point the vehicle's front to the right as shown in the diagram. Push the left or right arrow to make the wheels turn. The vehicle will drive forward if the wheels turns clock-wise.

Value: 20

⚠ If the wheels turn counter clock-wise, toggle on Reverse direction.

Reverse direction

Cancel Previous Next

Note

Vehicles distributed at AWS re:Invent 2018 might have their forward direction set in reverse. In such a case, make sure to toggle **Reverse direction**.

- d. To calibrate the maximum forward speed, under **Maximum forward speed**, gently move the slider left or right to adjust the **Maximum forward speed value** number gradually to such a positive value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Next**.

Calibration > Calibrate speed

Step 1 Raise your vehicle

Step 2 Calibrate stopped speed

Step 3 Set forward direction

Step 4 Calibrate maximum forward speed

Step 5 Calibrate maximum backward speed

Calibrate speed

Maximum forward speed

Move the slider to set the maximum forward speed on the vehicle so that the **Estimated speed** value matches, precisely or approximately, the value specified in training the model that is or will be loaded to the vehicle's inference engine.

Maximum forward speed value: 36

Slow Normal Fast Turbo Dangerous

Estimated speed:

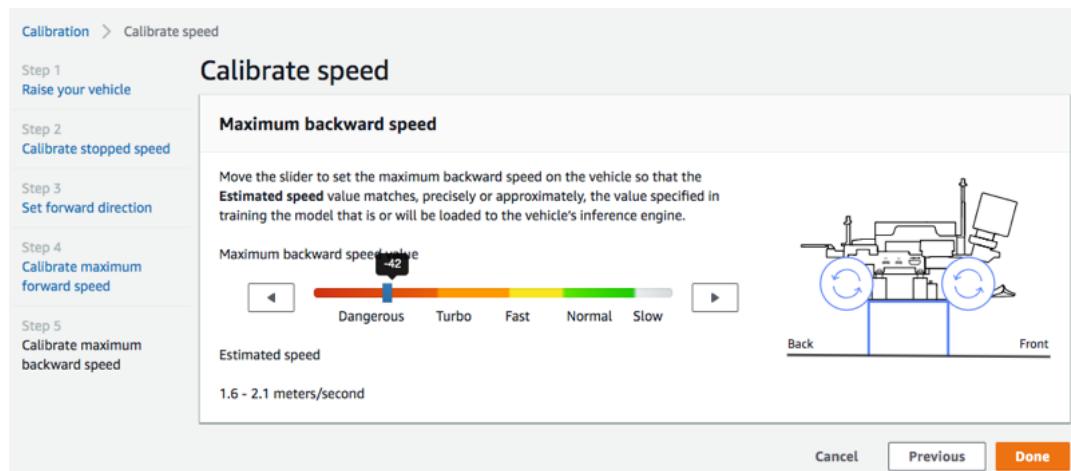
1.6 - 2.1 meters/second

Cancel Previous Next

Note

The actual maximum speed your vehicle depends on the friction of the track surface as well as the vehicle battery level. To make it flexible, you can set the vehicle's throttle limit to be 20-30 percent higher than the maximum speed specified for training in the simulation. Generally speaking, you should set the maximum speed value within the green area. Above that, your vehicle is likely to drive too fast at increased risk of breaking. Additionally, the action space for training doesn't support the maximum speed of more than 2 m/s.

- e. To calibrate the maximum backward speed, under **Maximum backward speed**, gently move the slider left or right to adjust the **Maximum backward speed value** number gradually to such a negative value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Done**.



Note

The AWS DeepRacer vehicle doesn't use backward speed in the autonomous driving mode. You can set the backward speed to any value with which you can comfortably control the vehicle's manual driving mode.

This concludes calibrating your AWS DeepRacer vehicle's maximum speed.

Upload a Model to Your AWS DeepRacer Vehicle

To start your AWS DeepRacer vehicle on autonomous driving, you must have uploaded at least one AWS DeepRacer model to your AWS DeepRacer vehicle.

To upload a model, you must have [trained and evaluated the model \(p. 28\)](#). You can train the model using the AWS DeepRacer console. After that, you need to download the model artifacts from its Amazon S3 storage to a (local or network) drive that can be accessed by your computer.

To upload a trained model to your vehicle

- Choose **Models** from the device console's main navigation pane.

Name	Size	Upload time	Status
model_22	34M	April 22, 2019, 9:14 AM PDT	Ready
Sample_Model	34M	April 22, 2019, 11:57 AM PDT	Ready

- On the **Models** page, choose **Upload** above the **Models** list.
- From the file picker, navigate to the drive or share where you've downloaded your model artifacts and choose the model for upload.

When the model is uploaded successfully, it will be added to the **Models** list and can be loaded into the vehicle's inference engine.

Drive Your AWS DeepRacer Vehicle

After [setting up your AWS DeepRacer vehicle \(p. 81\)](#), you can start to drive your vehicle manually or let it drive autonomously, using the vehicle's device console.

For autonomous driving, you must have trained an AWS DeepRacer model and have the trained model artifacts deployed to the vehicle. In the autonomous racing mode, the model running in the inference engine controls the vehicle's driving directions and speed. Without a trained model downloaded to the vehicle, you can use the vehicle's device console to drive the vehicle manually.

Many factors affect the vehicle's performance in autonomous driving. They include the trained model, vehicle calibration, track conditions, such as surface frictions, color contrasts and light reflections, etc. For your vehicle to achieve an optimal performance, you must make sure that the model transfer from the simulation to the real world is as accurate, relevant and meaningful. For more information, see [the section called "Optimize Training for Real Environments" \(p. 40\)](#).

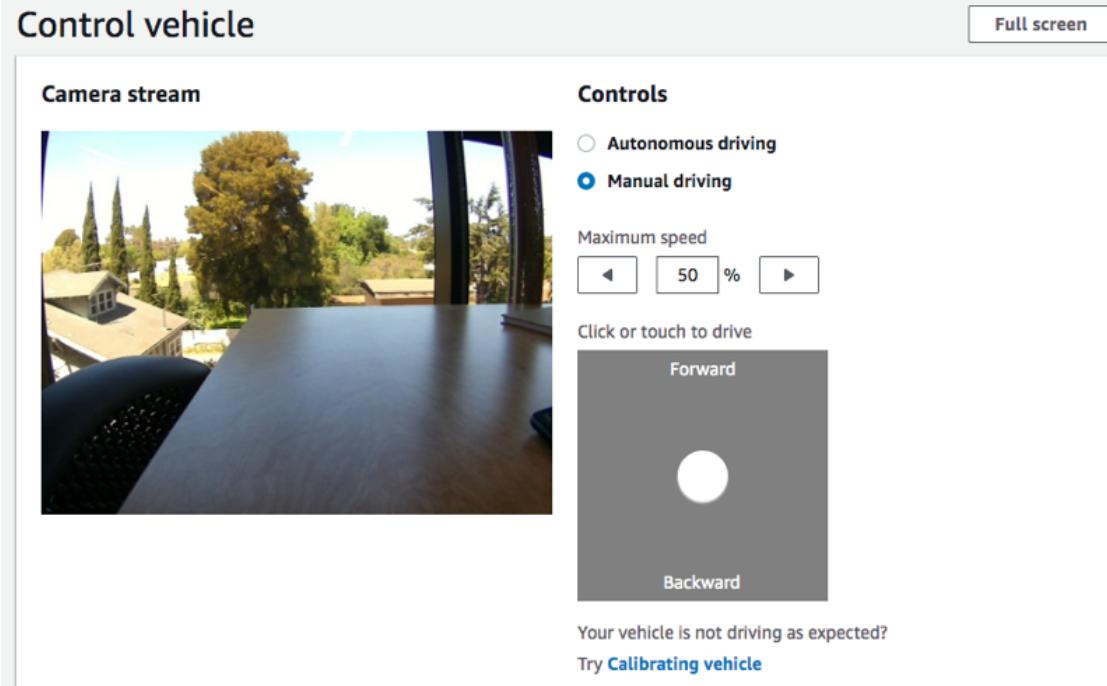
Drive Your AWS DeepRacer Vehicle Manually

If you have not trained any model or have not deployed any trained model to your AWS DeepRacer vehicle, you can't let it drive itself. But you can drive it manually.

To drive a AWS DeepRacer vehicle manually, follow the steps below.

To drive your AWS DeepRacer vehicle manually

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow [the instructions \(p. 83\)](#) to sign in to the vehicle's device control console.
2. On the **Control vehicle** page, choose **Manual driving** under **Controls**.



3. Under **Click or touch to drive**, click or touch a position within the driving pad to drive the vehicle. Images captured from the vehicle's front camera are displayed in the video player under **Camera stream**.
4. Repeat **Step 3** to drive the vehicle to different locations.

Drive Your AWS DeepRacer Vehicle Autonomously

To start autonomous driving, place the vehicle on a physical track and do the following:

To drive your AWS DeepRacer vehicle autonomously

1. Follow [the instructions \(p. 83\)](#) to sign in to the vehicle's device console, and then do the following for autonomous driving:
2. On the **Control vehicle** page, choose **Autonomous driving** under **Controls**.

Controls

Autonomous driving

Manual driving

Select a model

Select a model

Maximum speed

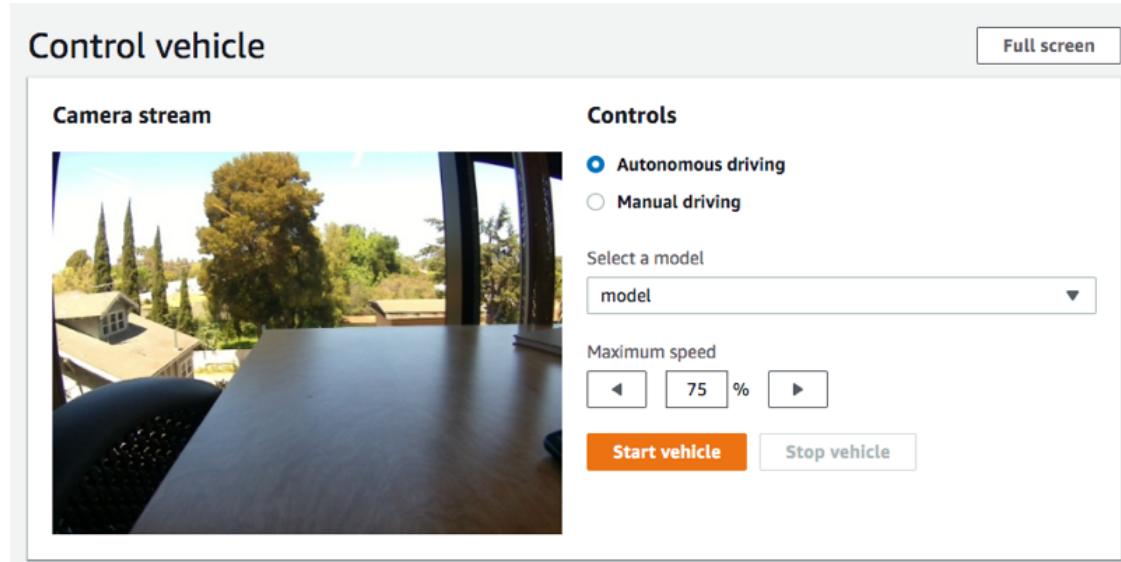
◀ 0 ▶

Start vehicle

Stop vehicle

-
3. From the **Select a model** drop-down list, choose an uploaded model. Then choose **Load model**. This will start loading the model into the inference engine. The process takes about 10 seconds to complete.
 4. Adjust the **Maximum speed** setting of the vehicle to be a percentage of the maximum speed used in training the model.

Certain factors, such as surface friction of the real track, can reduce the maximum speed of the vehicle from the maximum speed used in the training. You'll need to experiment to find the optimal setting.



5. Choose **Start vehicle** to set the vehicle to drive autonomously.
6. Watch the vehicle drive on the physical track or the streaming video player on the device console.
7. To stop the vehicle, choose **Stop vehicle**.

Repeat from Step 3 for another run with the same or a different model.

Manage Your AWS DeepRacer Vehicle Settings

After the initial setup, you can use the AWS DeepRacer device control console to manage your vehicle's settings. The tasks include the following:

- choosing another Wi-Fi network,
- resetting the device console password,
- enabling or disabling the device SSH settings.
- configuring the vehicle rear LED color.

The procedure below walks you through these tasks.

To manage your vehicle's settings

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow [the instructions \(p. 83\)](#) to sign in to the vehicle's device control console.
2. Choose **Settings** from the main navigation pane.
3. On the **Settings** page, perform one or more of the following tasks of your choosing.

The screenshot shows the 'AWS DeepRacer: Vehicle' settings interface. On the left, a sidebar lists options like Control Vehicle, Models, Calibration, Settings (which is selected), and Logs. Below this are links for Build a track, Train a model, and two IP addresses (IP: 10.92.206.61 and IP: 192.168.9.194). A Logout button is also present. The main content area is titled 'Settings' and contains four sections: 'Network settings', 'Device console password', 'Device SSH', and 'LED color'. Each section has an 'Edit' button. The 'Network settings' section shows a Wi-Fi network SSID of 'Mobile' and an IP address of '10.92.206.61, 192.168.9.194'. The 'Device console password' section shows a masked password. The 'Device SSH' section shows an SSH server status of 'Enabled' and a masked password. The 'LED color' section shows a color swatch set to 'Green'.

- a. To choose another Wi-Fi network, choose **Edit** for **Network settings** and then follow the steps below.
 - i. Follow the instructions, shown on **Edit network settings**, to connect your vehicle to your computer using the USB-to-USB-C cable. After the **USB connection** status becomes **Connected**, choose the **Go to deepracer.aws** button to open the device console login page.

Settings > Edit network settings

Edit network settings

Network settings

Wi-Fi network SSID Mobile	IP address 10.92.206.61, 192.168.9.194	USB connection Not connected
------------------------------	---	---------------------------------

Instructions

1. Connect your vehicle to your computer.

Use the included USB cable to connect your computer to the vehicle



- ii. On the device console login page, type the password printed on the bottom of your vehicle and then choose **Access vehicle**.
- iii. Under **Wi-Fi network details**, choose a Wi-Fi network from the drop-down list, type the password of the chosen network, and then choose **Connect**.

Wi-Fi network details

Specify your Wi-Fi network details.

Wi-Fi network name (SSID)
ATT807

Wi-Fi password
••••••
 Show password

Connect

- iv. After the **Vehicle status** for the Wi-Fi connection becomes **Connected**, choose **Next** to return to the **Settings** page of the device console, where you'll see a new IP address of the vehicle.
- b. To reset the password for signing in to the device console, choose **Edit for Device console password** and then follow the steps below.
 - i. On **Edit device console password** page, type a new password in **New password**.
 - ii. Retype the new password in **Confirm password** to confirm your intention for the change. The password value must be the same before you can move on.
 - iii. Choose **Change password** to complete the task. This option is activated only if you have entered and confirmed a valid password value in the steps above.

Settings > Edit device console password

Edit device console password

You are required to setup a password to protect access to your AWS DeepRacer vehicle. If you forget your password, [reset your password](#).

Old password

New password

Confirm password

|

Show passwords

Change password

- c. To enable or disable SSH connection to the vehicle, choose **Edit for Device SSH** and then choose **Enable or Disable**.

Settings > Edit device SSH

Edit device SSH

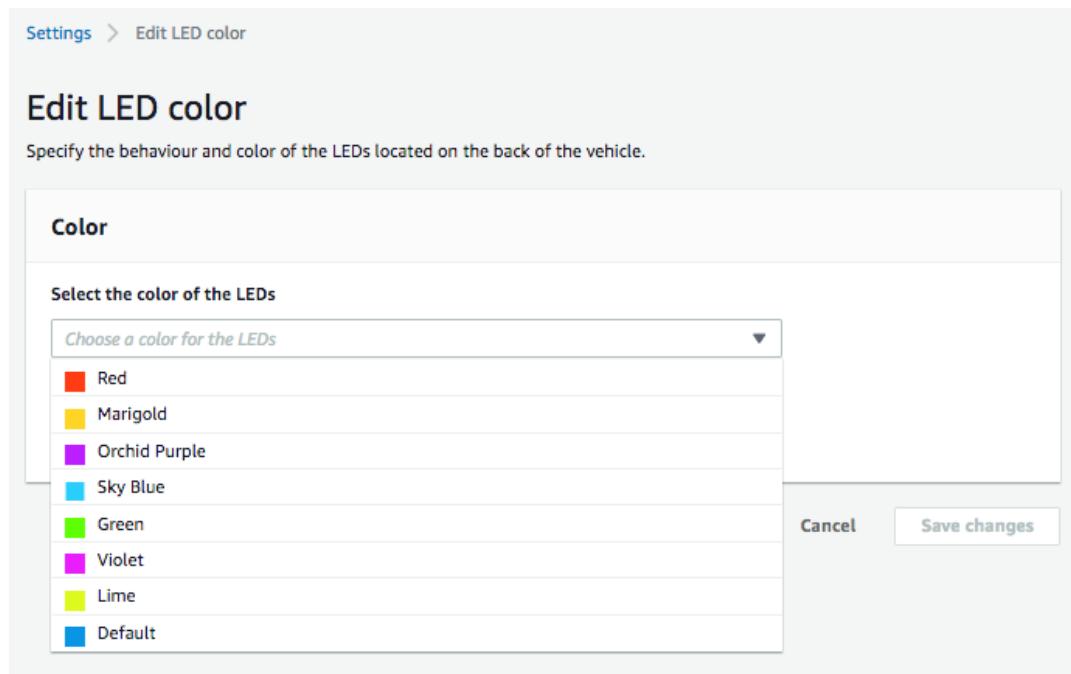
SSH Server

Enable the SSH server on your device to enable login via CLI to execute command

Disabled Enabled

ⓘ Certain device functions such as software update are not supported over SSH.

4. To change the vehicle's rear LED color as a form of identification of the vehicle, choose **Edit for LED color** on the **Settings** page and do the following.
- Choose an available color from the **Select the color of the LEDs** drop-down list on the **Edit LED color** page.



You should choose a color that can help identify your vehicle from other vehicles sharing the track at the same time.

- b. Choose **Save changes** to complete the task.

The **Save changes** functionality becomes active only after you have chosen a color.

View Your AWS DeepRacer Vehicle Logs

Your AWS DeepRacer vehicle logs operational events that can be helpful for troubleshooting issues encountered in running your vehicle. There are two types of AWS DeepRacer vehicle logs:

- The system event log keeps track of operations taking place in the vehicle's computer operating system, such as process managing, Wi-Fi connecting or password reset events.
- The robot operating system logs record statuses of operations taking place in the vehicle's operating system node for robotic operations, including vehicle driving, video streaming and policy inferencing operations.

To view the device logs, follow the steps below.

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow [the instructions \(p. 83\)](#) to sign in to the vehicle's device control console.
2. Choose **Logs** from the device console's main navigation pane.
3. To view the system events, scroll down the event list under **System event log**.

System event log

```
Apr 8 15:16:07 amss-42im login: message repeated 2 times: [ <INFO> Status returned from login proxy: 200]
Apr 8 15:16:07 amss-42im wifi_settings: <INFO> Check OTG Link State: not connected
Apr 8 15:16:07 amss-42im wifi_settings: <INFO> host: https://10.92.206.61/home otg_connected: not connected is_usb_connected: not connected
Apr 8 15:16:07 amss-42im login: <INFO> Status returned from login proxy: 200
Apr 8 15:16:07 amss-42im login: message repeated 2 times: [ <INFO> Status returned from login proxy: 200]
Apr 8 15:16:07 amss-42im vehicle_control: <INFO> Changed the vehicle state to auto
Apr 8 15:16:07 amss-42im login: <INFO> Status returned from login proxy: 200
Apr 8 15:16:07 amss-42im wifi_settings: <INFO> Check OTG Link State: not connected
Apr 8 15:16:08 amss-42im utility: <INFO> Command executing: hostname -
Apr 8 15:16:08 amss-42im utility: <INFO> ['10.92.206.61 192.168.9.194', '']
Apr 8 15:16:11 amss-42im login: <INFO> Status returned from login proxy: 200
Apr 8 15:16:41 amss-42im login: message repeated 3 times: [ <INFO> Status returned from login proxy: 200]
Apr 8 15:16:41 amss-42im ssh_api: <INFO> Providing ssh enabled as response
Apr 8 15:16:41 amss-42im utility: <INFO> Command executing: /bin/systemctl --no-pager status ssh
Apr 8 15:16:41 amss-42im wifi_settings: <INFO> Check OTG Link State: not connected
Apr 8 15:16:41 amss-42im utility: <INFO> * ssh.service - OpenBSD Secure Shell server#012 Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled) #012 Active: active (running) since Fri 2019-04-05 15:43:20 EDT; 2 days ago#012 Main PID: 16466 (sshd)#012
CGroup: /custom_elica/etc/service#012 └─16466 /usr/bin/sshd -D#012 Apr 08 14:27:07 amss-42im sshd[11706]: Accepted password for

```

4. To view the robot operating system events, scroll down the event list under **Robot operating system log**.

Robot operating system log

```
1554750920.064320544 Node Startup
1554750920.131309136 INFO [/opt/workspace/AwsSilverstoneDeviceLib/ros-src/servo_pkg/src/servo_node.cpp:439(LedMgr::LedMgr) [topics: /rosout] LedMgr pwm channel creation
1554750920.201161384 INFO [/tmp/binarydeb/ros-kinetic-roscpp-1.12.14/src/libros/service.cpp:80(service::exists) [topics: /rosout] waitForService: Service [/media_state] has not been advertised, waiting...
1554750920.640698003 INFO [/tmp/binarydeb/ros-kinetic-roscpp-1.12.14/src/libros/service.cpp:122(service::waitForService) [topics: /rosout] waitForService: Service [/media_state] is now available.
1554750920.578106989 INFO [/opt/workspace/AwsSilverstoneDeviceLib/ros-src/web_video_server /src/web_video_server.cpp:96(WebVideoServer::spin) [topics: /rosout] Waiting For connections on 0.0.0.0:8080
1554750921.752294063 INFO [navigation_node.py:154(set_action_space_scales) [topics: /auto_drive, /rosout, /rl_results] Action space scale set: {'steering_max': 30.0, 'speed_max': 0.8}
Mapping equation params a: -1.875 b: 2.75
1554750930.167246103 INFO [software_update_process.py:25(logger) [topics: /rosout] /software_update: [04/08/19 15:15:30] Setup Ethernet over OTG.
1554750930.174333095 INFO [software_update_process.py:25(logger) [topics: /rosout] /software_update: [04/08/19 15:15:30] Entering daemon loop.
1554750930.205965042 INFO [software_update_process.py:25(logger) [topics: /rosout] /software_update: [04/08/19 15:15:30] Updating network information.
1554750930.209075927 INFO [software_update_process.py:25(logger) [topics: /rosout] /software_update: [04/08/19 15:15:30] Checking software update...
1554750938.287539958 INFO [software_update_process.py:25(logger) [topics: /rosout] /software_update: [04/08/19 15:15:38] Verifying package aws-deepracer-core...
```

Build Your Physical Track for AWS DeepRacer

This section describes how you can build a physical track for a AWS DeepRacer model. To drive your AWS DeepRacer autonomously and to test your reinforcement learning model in a physical environment, you need a physical track. Your track resembles the simulated track used in training and replicates the environment used to train the deployed AWS DeepRacer model.

Topics

- [Track Materials and Build Tools \(p. 100\)](#)
- [Lay Your Track for AWS DeepRacer \(p. 101\)](#)
- [AWS DeepRacer Track Design Templates \(p. 105\)](#)

Track Materials and Build Tools

Before you start to construct you track, get the following materials and tools ready.

Topics

- [Materials You May Need \(p. 100\)](#)
- [Tools You May Need \(p. 100\)](#)

Materials You May Need

To build a track, you need the following materials:

- For track borders:

You can create a track with tape that is about 2-inches wide and white or off-white color against the dark-colored track surface. For a dark surface, use a white or off-white tape. For example, [1.88 inch width, pearl white duct tape](#) or [1.88 inch \(less sticky\) masking tape](#).

- For track surface:

You can create a track on a dark-colored hard floor such as hardwood, carpet, concrete, or [asphalt felt](#). The latter mimics the real-world road surface with minimal reflection. [Interlocked foam](#) or rubber pads are also good options.

Tools You May Need

The following tools are either required or helpful to design and build your track:

- Tape measure and scissors

A good tape measure and a pair of scissors are essential for building your track. If you don't already have one, you can order [a tape measure here](#) or [scissors here](#).

- Optional design tools

To design your own track, you might need a [protractor](#), a [ruler](#), a [pencil](#), a [knife](#) and a [compass](#).

Lay Your Track for AWS DeepRacer

When you build your track, it's a good practice to start with a simple design, such as a straight or single-turn track. Next you can move on to looped tracks. Here, we use a single-turn track as an example to walk you through the steps to construct your own track. First let's review dimensional requirements of a track.

Topics

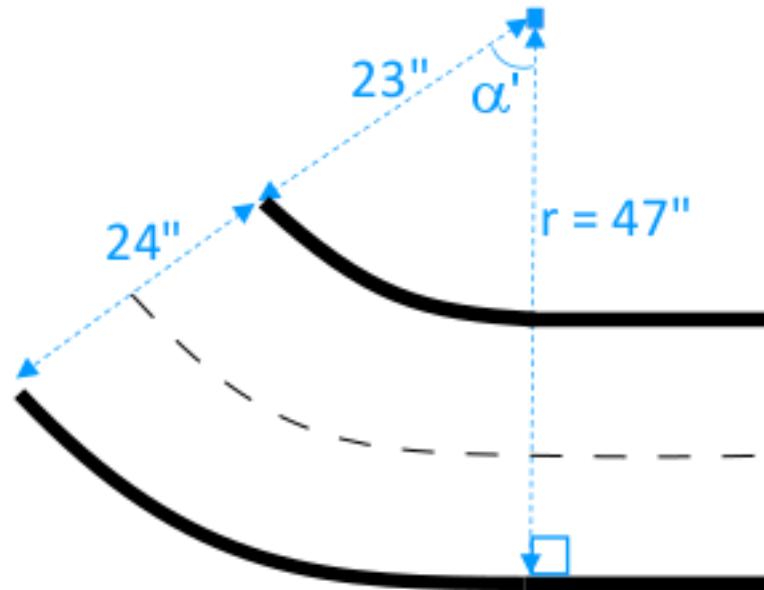
- Dimensional Requirements (p. 101)
- Considerations for Model Performance (p. 102)
- Steps to Build the Track (p. 102)

Dimensional Requirements

You can build a track of any shape as long as it meets the following requirements:

- **Minimum turning radius:**

On a curved track, the turning radius (r) measures from the circle center to the outside border, as illustrated below.



The minimum turning radius (r_{\min}) depends on the track turning angle (α) at a corner and should comply to the following limits:

- If the track's turning angle is $\# \leq 90$ degrees,

$r_{\min} \geq 25$ inches

We recommend 30 inches.

- If the track's turning angle is $\# > 90$ degrees, α

$r_{\min} \geq 30$ inches.

We recommend 35 inches.

- **Track width,**

The track width (w_{track}) should comply to the following limit:

$w_{track} \geq 24 \pm 3$ inches.

- **Track surface:**

The track surface should be smooth and of a uniform dark color. The minimum enclosing area should be 30 inches x 60 inches in size.

Carpeted and wood floors work well. [Interlocked foam or rubber pads](#) match the simulated environment better than wood, but this is not required. Concrete floors can be problematic due to light reflection on the surface.

- **Track barrier**

Though not required, we recommended that you encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

Considerations for Model Performance

How you build a track can affect the reliability and performance of a trained model. The following are factors you should consider when building your own tracks.

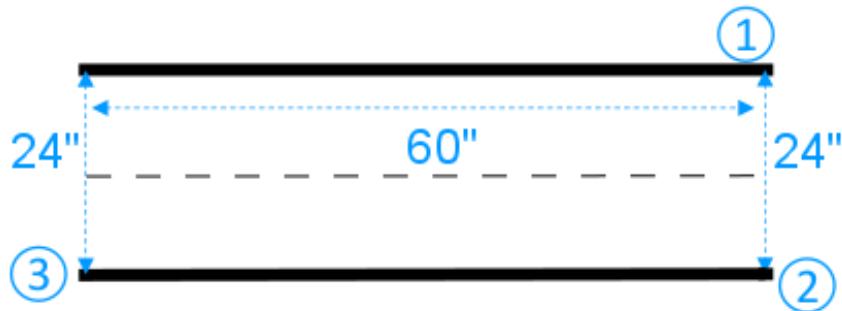
1. Do not place any white objects on or near your track. If necessary, remove any white object from the track or its vicinity. This is because training in the simulated environment assumes that only the track borders are white.
2. Use clean and continuous tape to mark the track borders. Broken or creased track borders can affect the trained model performance.
3. Avoid using a reflective surface as the track floor. Reduce glare from bright lights. The glare from straight edges can be misinterpreted as objects or borders.
4. Do not use a track floor with line markings other than the track lines. The model might interpret the non-track lines as part of the track.
5. Place barriers around the track to help reduce distractions from background objects.

Steps to Build the Track

As an illustration, we use the most basic single-turn track. You can modify the instructions to create a more complex track such as an S-curve, a loop, or the AWS re:invent 2018 track.

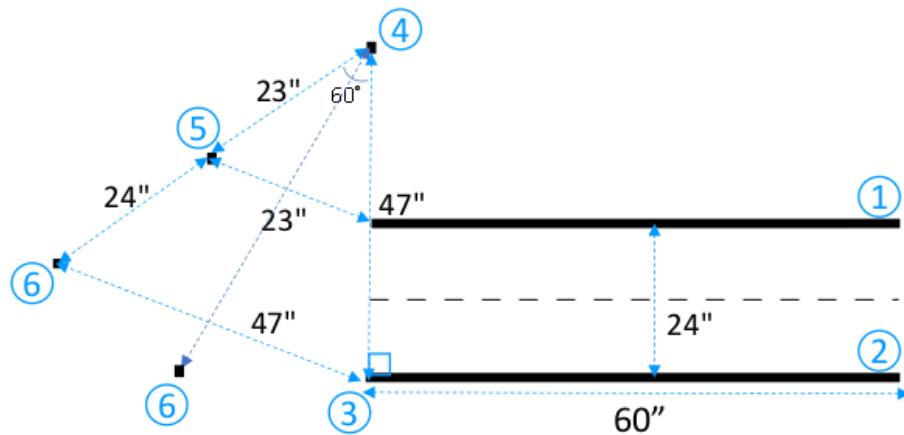
To build an AWS DeepRacer single-turn track

1. To construct the straight portion of the track, follow the steps below and refer to the diagram.
 - a. Put a 60-inch long piece of tape on the floor to lay down the first border in a straight line (1).
 - b. Use a tape measure to locate the second border's two end points, (2) and (3). Put them 24 inches apart from the first border's two ends.
 - c. Put another 60-inch long piece of tape on the floor to lay down the second boarder to connect the two endpoints (2) and (3).



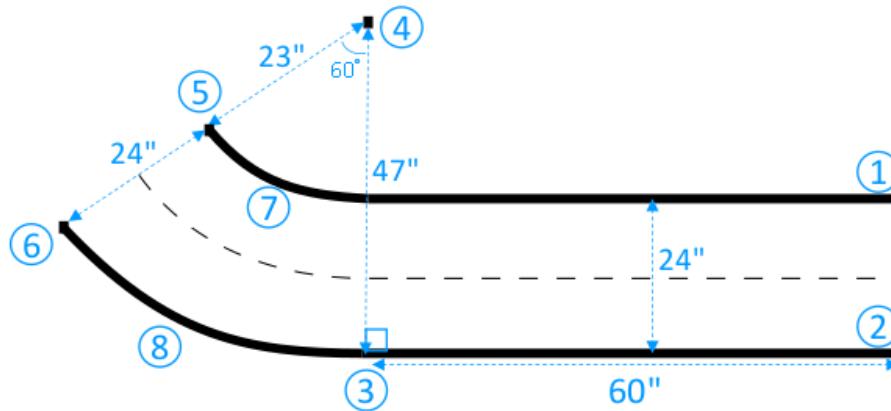
We assume the straight track segment is 60-inches long and 24-inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met.

2. To make the track to turn at a 60-degree angle, do the following and refer to the diagram:
 - a. Use the tape measure to locate the center (4) of the turning radius (4-3 or 4-6). Mark the center with a piece of tape.
 - b. Draw an equilateral triangle. The three sides are (3-4), (4-6), and (6-3).



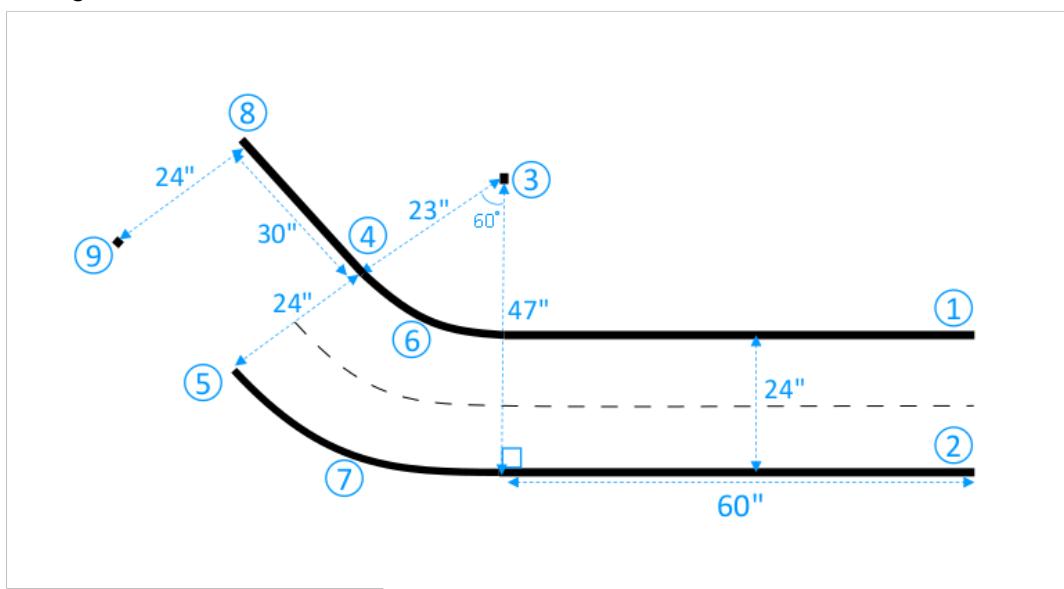
To make a 60-degree turn along the track, use the equilateral triangle (3-4-6) to determine the locations of the two final end points (5) and (6) for the curved track segment. For turns at a different angle, you can use a protractor (or a protractor app) to locate the two final ends (5) and (6) of the curved track segment. Turning radius variations are acceptable as long as the minimum turning radius requirement in Step 2 is met.

- c. Put small tape segments, e.g. 4-inches each, on the floor to lay the curved border segments (7) and (8) and connect them with the straight-line borders. The two curved borders don't need to be parallel.



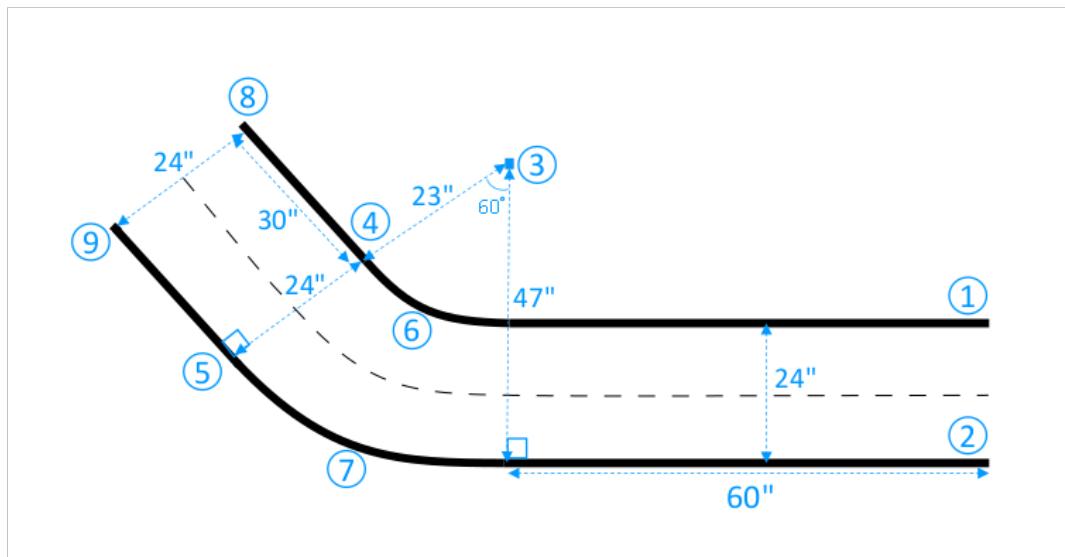
- 3. To extend the track with the next straight segment of 30 inches long and 24 inches wide, do the following:

- a. Put a 30-inch long piece of tape on the floor to lay down the first border (4-8) perpendicular to the edge (3-5).



- b. Use the tape measure to locate the ending point of the second border (9). You can customize the length of the straight lines to fit to the space you have.

- c. Put another 30-inch long piece of tape on the floor to lay down the second border (5-9) perpendicular to the edge (3-5).



We assume the second straight track segment is 30 inches long and 24 inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met and the dimensions are consistent with other track segments.

4. Optionally, cut tape segments of 4 inches long and then place the tape segments 2 inches apart along the track center to lay the dashed center line.

You've now finished building the single-turn track. To help your vehicle to better distinguish the drivable surfaces from non-drivable surfaces, you should paint the off-track surface a color of sufficient contrast with respect to the on-track surface color. To ensure safety, you could encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

You can apply the instructions to extend the track to [more complex shapes \(p. 105\)](#).

AWS DeepRacer Track Design Templates

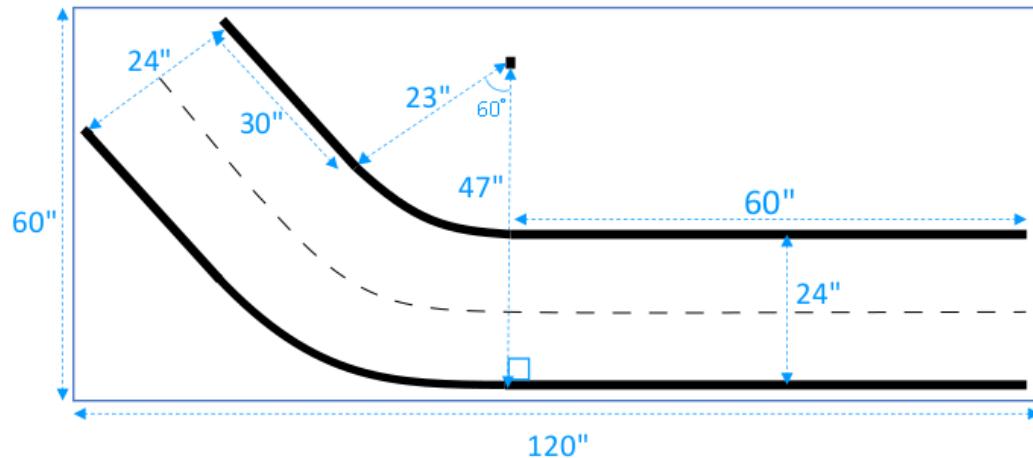
The following track design templates show AWS DeepRacer tracks that you can build by following the [instructions \(p. 101\)](#) presented in this section.

Topics

- [AWS DeepRacer Single-Turn Track Template \(p. 105\)](#)
- [AWS DeepRacer S-Curve Track Template \(p. 106\)](#)
- [AWS DeepRacer Loop Track Template \(p. 106\)](#)
- [AWS DeepRacer AWS re:Invent 2018 Track Template \(p. 107\)](#)

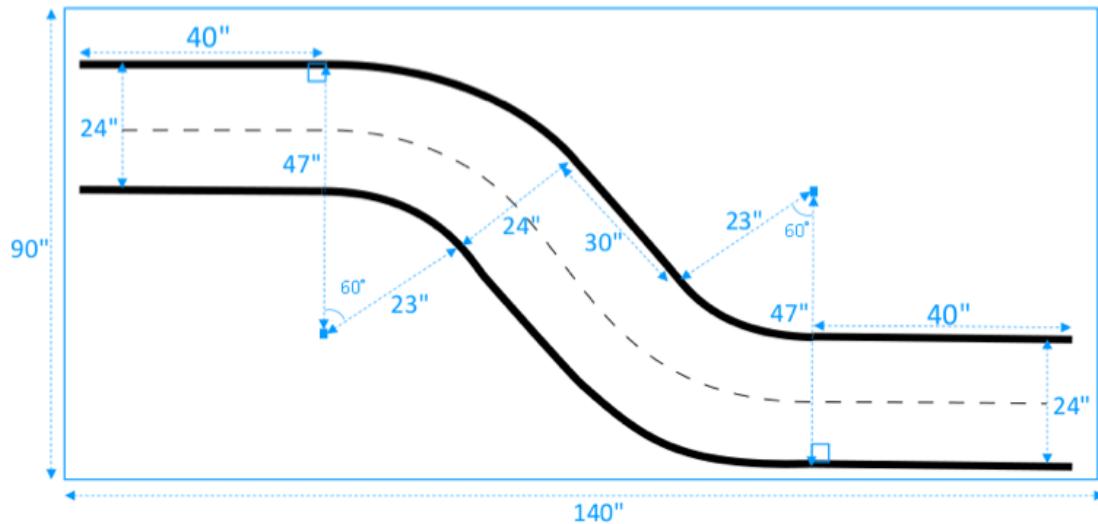
AWS DeepRacer Single-Turn Track Template

This basic track template consists of two straight track segments connected by a curved track segment. Models trained with this track should make your AWS DeepRacer vehicle drive in straight line or to make turns in one direction.



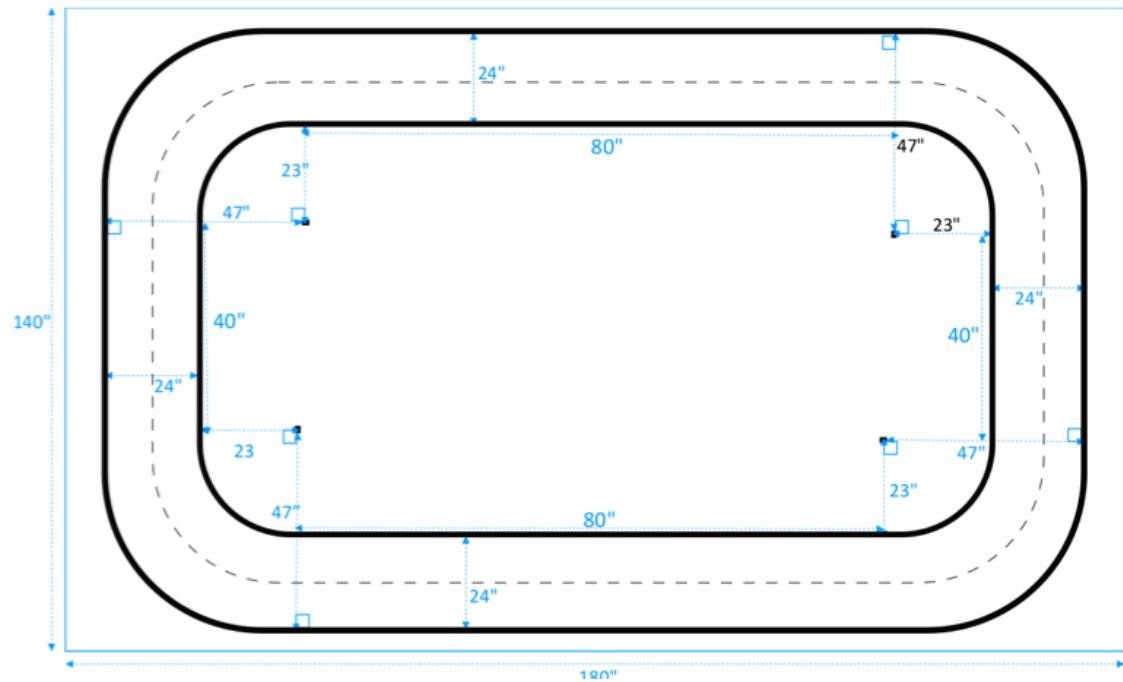
AWS DeepRacer S-Curve Track Template

The track is more complex than the single-turn track because the model needs to learn to make turns in two directions. You can easily extend the single-turn track construction instructions to this track by turning it in the opposite direction after the first turn.



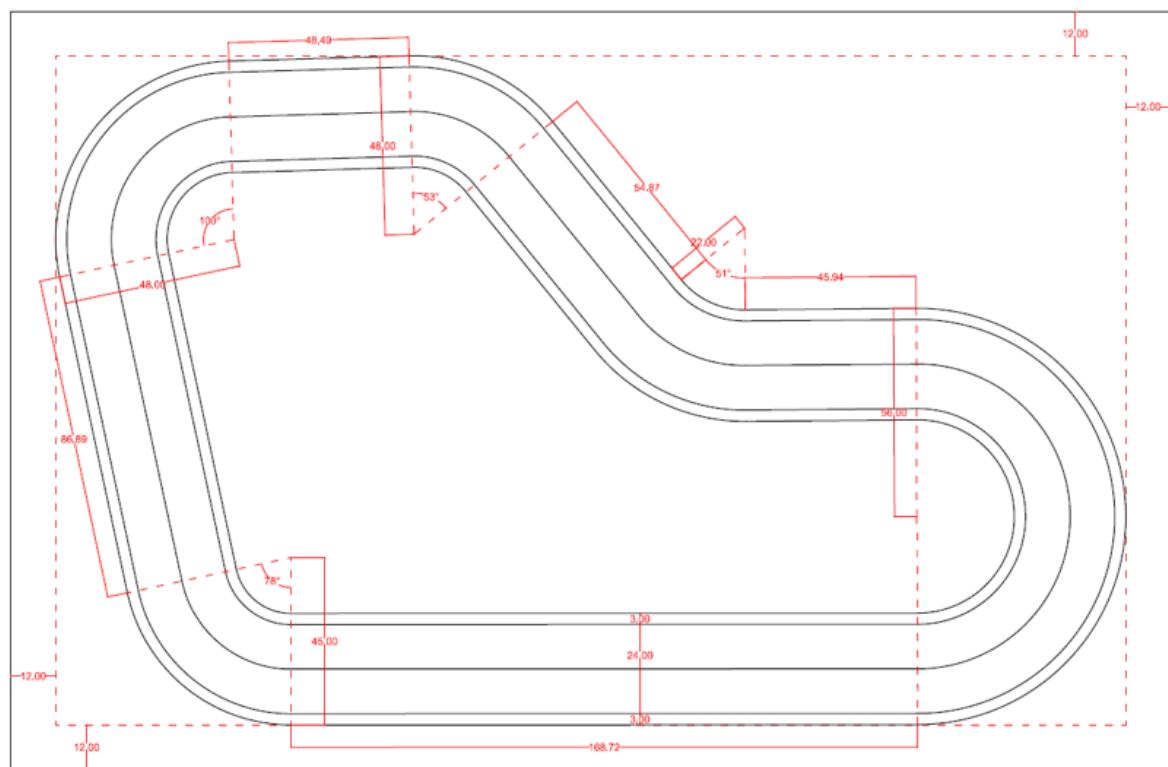
AWS DeepRacer Loop Track Template

This regular loop track is a repeating, 90-degree, single-turn track. Obviously, it requires a larger enclosing area for laying the entire track.



AWS DeepRacer AWS re:Invent 2018 Track Template

This less regular loop track was first presented at the AWS re:Invent 2018. Models trained on this track must learn how to turn left and right at various angles and how to move straight ahead in various directions.



At AWS re:Invent 2018, the track size measured 26 feet long by 17 feet wide. To change the scale, be aware that the scaled-down track may become too narrow to fit the model trained on a wider track.

To reproduce the same color production, use the following color specifications:

- Green: PMS 3395C
- Orange: PMS 137C
- Black: PMS 432C
- White: CMYK 0-0-2-9

This track was tested with the following materials for its surface:

- Carpet

The track was printed on 8-ounce, dye-sublimated, polyester-faced carpet with latex rubberized backing. Carpet is durable, provides great performance, but is expensive.

- Vinyl

The track was printed on 13-ounce scrim vinyl with a matte finish to reduce glare. Vinyl is typically cheaper than carpet and provides good performance. Vinyl is not as durable as carpet.

Due to its large size, the track cannot be easily printed on a single piece of material. Align track lines well when connecting pieces together.

For more detailed information, see [the AWS DeepRacer re:Invent 2018 track specification](#).

Rank AWS DeepRacer Models in Leaderboard

After successfully tuning and evaluating your model in simulations, you may want to compare your model's performance to other user's models. The comparison provides another measure of the fitness of your model. To support this, you can submit your model to an AWS DeepRacer leaderboard and see your model ranked against other participant models.

Each AWS DeepRacer leaderboard is associated with a racing event that uses a specific track. The leaderboard manages a list of performance rank for user-submitted AWS DeepRacer models and contains user aliases and average lap times of the included models. The performance of a model is ranked according to the average lap time over a number of consecutive successful trials. Each leaderboard can specify between three and five consecutive laps for performance ranking. If a model evaluation does not result in the specified number of consecutive successful trials, the model is not included in the leaderboard.

There are two types of racing events in AWS DeepRacer League. One is the DeepRacer League Virtual circuit and the other the DeepRacer League Summit circuit. The Virtual circuit consists of a series of virtual races. In a virtual race, you compete against other participants on a virtual track and qualified results are ranked in the associated virtual leaderboard. The Summit circuit consists of a series of live, real-time racing events. In a live race, you compete against other participants in person on a physical track. Qualified results are ranked in the associated physical leaderboard.

You can use the AWS DeepRacer console to enter a virtual race and submit a trained model for timed trials in the simulator. This is how your qualified results get ranked in the virtual leaderboard. New virtual racing events are opened as old ones close. For physical races, you must enter in person and start your AWS DeepRacer vehicle for autonomous driving on the physical track. The qualified result is posted to the associated leaderboard through the vehicle's Wi-Fi connection. A physical race leaderboard is typically provided by a third-party vendor. For example, [GoSquared](#) provided the physical leaderboard for the *AWS re:Invent 2018* racing event. Currently, all AWS DeepRacer leaderboards are public and open to any AWS DeepRacer user.

In this section, we discuss AWS DeepRacer virtual leaderboards exclusively.

Topics

- [Submit Your Model to an AWS DeepRacer Leaderboard \(p. 109\)](#)

Submit Your Model to an AWS DeepRacer Leaderboard

In this section, you can see the steps to use the AWS DeepRacer console to submit your trained model to a virtual race.

To submit your model to a virtual leaderboard in the Virtual circuit

1. Sign in to the AWS DeepRacer console at <https://console.aws.amazon.com/deepracer>.
2. From the main navigation pane, choose **DeepRacer Virtual Circuit**.

AWS DeepRacer > DeepRacer Virtual Circuit

Welcome me

AWS DeepRacer League

Welcome to the world's first global autonomous racing league, open to anyone. It's time to race for prizes, glory, and a chance to advance to the AWS DeepRacer Championship Cup at re:invent 2019 to win the coveted AWS DeepRacer Cup. Get on the track to compete in the live events at 20 AWS Summits worldwide or enter the monthly virtual races.

Summit circuits

Get ready to race in the Virtual Circuit

- 1 **Pick a race**
The virtual races will take place
- 2 **Train a model**
Whether you are new to machine
- 3 **Win prizes**
Take part in multiple races to

3. On the DeepRacer Virtual Circuit page, choose an open race, and then choose Submit model.

AWS DeepRacer > DeepRacer Virtual Circuit

1. Pick a race

The virtual races will take place monthly on new competition tracks and are open to all levels of expertise. Select the race below to get more information.

2. Train a model

Whether you are new to machine learning or ready to build on your existing skills, we can help you get ready to race.

Get started

3. Win prizes

Take part in multiple races to earn points. Winners advance to the . Knockout round

[Terms and Conditions](#)

Summit Speedway Pre-season

Open 5d 20h 02m remaining

Race information Submit model

4. If this is your first time participating in an AWS DeepRacer League racing event, set your alias in **Racer name** under **AWS DeepRacer League alias**.

AWS DeepRacer League alias

Racer name

Choose a unique alias to be used for all Virtual Circuit races. If your model ranks, your alias will be used on the race leaderboard.

your-alias

Must be between 2 and 128 characters and only contain letters, numbers and - (hyphen). No spaces or underscores.

5. Under **Model submission details**, choose your model from the **Model** drop-down menu.

AWS DeepRacer > DeepRacer Virtual Circuit > Summit Speedway Pre-season >
submit model

Submit model to Summit Speedway Pre-season

Model submission details

Virtual Race

Summit Speedway Pre-season



Time remaining

05 19 00

Day Hour Minute

Take part in the **Summit Speedway**, the pre-season race for the Virtual Circuit. Train your model on the re:Invent track to get the best chance of posting a good lap time on this track. You can also use this model to race at any of the upcoming **Summit Circuit** races.

Model

Submit your model to participate in the virtual race. Your model will be ranked based on the average time it takes to complete a lap on the race track in our virtual simulator. Your time and rank will be displayed in the race leaderboard alongside other competitors. You may only submit trained models.

trained-on-reinvent-2018-track-speed-limit-5



6. If this is your first time participating in a DeepRacer League event, under **User terms and services**, check the **I have read and agree to the terms and conditions of the DeepRacer League** option.

User terms and services

I have read and agree to the terms and conditions of the [DeepRacer League Agreement](#).

Follow the link to read the terms and conditions before checking this option.

7. Choose **Submit model** to complete the submission.

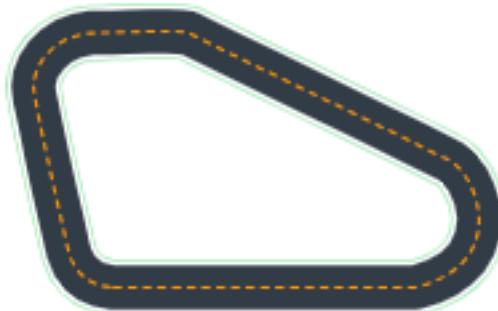
Cancel

Submit model

After your model is submitted, AWS DeepRacer starts to evaluate it. The process takes some time.

8. On the racing event page, inspect the race track to ensure that your model has been trained to handle the track shape.

Summit Speedway Pre-season



Your agent must complete 3 consecutive laps.

Submit model

Retrain your model with different training configurations and submit the newly trained model to improve its ranking in the leaderboard.

9. On the racing event page, inspect submission status under your alias.

Your rank	58
Racer name:	me
Your best average lap time	00:00:26.725
Your best model	trained-on-reinvent-2018-track-speed-limit-5
Latest model submitted	
	00m 13s until you can resubmit
Name	trained-on-reinvent-2018-track-speed-limit-5
Submission time	4/23/2019, 1:01:20 PM
Status	 Completed 3 laps
Average lap time	00:00:26.725

10. On the racing event page, inspect the ranking list under **Leaderboard** to see how your model compares with others.

Leaderboard (89)			
Rank	Racer	Average lap time	Submission Time
1	Nick-K	00:11.486	4/21/2019, 2:43:36 AM
2	Nick	00:11.555	4/19/2019, 4:43:36 AM
3	mc	00:11.705	4/19/2019, 8:32:12 PM
4	kazuyaM	00:11.723	4/18/2019, 11:00:12 PM
5	vinas	00:11.764	4/22/2019, 1:00:48 PM
6	JRacer	00:11.882	4/22/2019, 2:11:54 PM
7	bart	00:11.990	4/21/2019, 4:55:39 AM
8	fail-man	00:12.202	4/19/2019, 9:11:23 AM
9	cheungt	00:12.209	4/23/2019, 9:16:25 AM
10	MR	00:12.418	4/23/2019, 7:39:23 AM

If your model cannot finish the track in specified three consecutive trials, it is not included in the ranking list under **Leaderboard**. If a subsequent submission yields a faster result, your model's ranking is updated to reflect your best performance at the time.

Troubleshooting AWS DeepRacer Issues

Here you'll find troubleshooting tips for frequently asked questions as well as late-coming bug fixes.

Topics

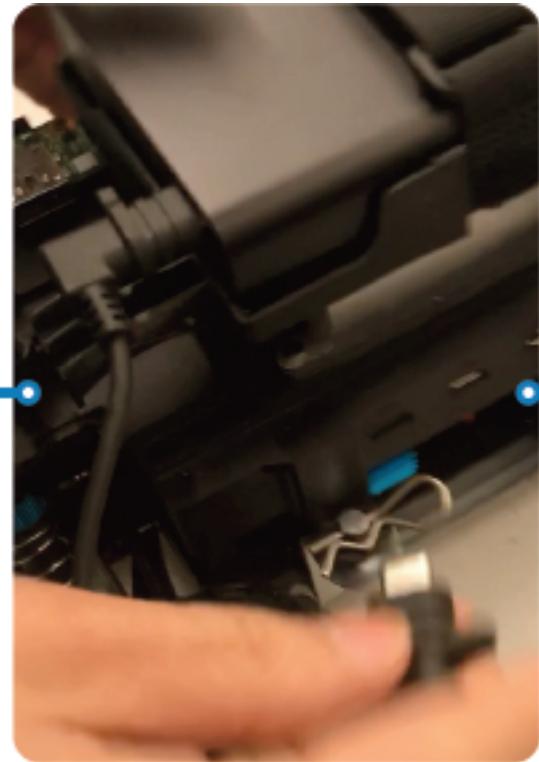
- [How to Switch AWS DeepRacer Compute Module Power Source from Battery to Power Outlet? \(p. 115\)](#)
- [How to Connect Your AWS DeepRacer to Your Wi-Fi Network? \(p. 117\)](#)
- [How to Charge the AWS DeepRacer Drive Module Battery? \(p. 117\)](#)
- [How to Charge the AWS DeepRacer Compute Module Battery? \(p. 118\)](#)
- [How to Maintain Vehicle's Wi-Fi Connection? \(p. 119\)](#)
- [How to Get the MAC Address of Your AWS DeepRacer Device? \(p. 120\)](#)
- [How to Recover Your AWS DeepRacer Device Console Default Password? \(p. 121\)](#)
- [How to Manually Update Your AWS DeepRacer Device? \(p. 122\)](#)
- [How to Restore Your AWS DeepRacer Vehicle to Factory Settings? \(p. 123\)](#)

How to Switch AWS DeepRacer Compute Module Power Source from Battery to Power Outlet?

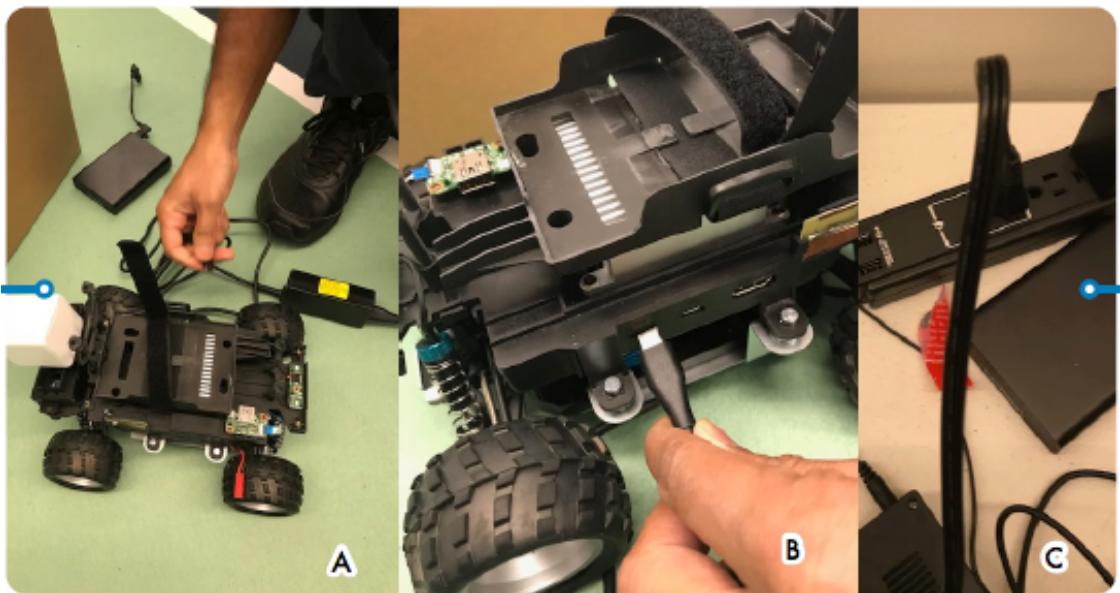
If the compute module battery level is low when you set up your AWS DeepRacer for the first time, follow the steps below to switch the compute power supply from the battery to a power outlet:

1. Unplug the USB-C cable from the vehicle's compute power port.

AWS DeepRacer Developer Guide
How to Switch AWS DeepRacer Compute Module
Power Source from Battery to Power Outlet?



2. Attach the AC power cord and the USB-C cable to the computer module power adapter (A). Plug the power cord to a power outlet (C) and plug the USB-C cable the vehicle's computer module power port (B).



How to Connect Your AWS DeepRacer to Your Wi-Fi Network?

To use your AWS DeepRacer, you must connect the vehicle to your home or office Wi-Fi network. To connect the vehicle to your Wi-Fi network, follow the steps below:

1. Have a USB flash drive on hand.
2. Plug in the USB flash drive to your computer.
3. Open a web browser on your computer and navigate to <https://d1.awsstatic.com/deepracer/wifi-creds.txt> to download the Wi-Fi configuration file and copy it to the USB drive.
4. Open the Wi-Fi configuration file in a text editor and type the name (SSID) and password of your Wi-Fi network in the corresponding fields.
5. Eject the USB drive from your computer and then plug it into the USB port on the back of the vehicle.



6. Watch the Wi-Fi LED on the vehicle to blink and then to turn blue. The vehicle is now connected to the Wi-Fi network. Unplug the USB drive and skip the next step.
7. If the Wi-Fi LED turns red after blinking, unplug the USB drive from the vehicle. Plug the USB drive back to your computer, verify that the configuration contains the correct network name and password, correct any mistakes or typos, save the file. Repeat Step 5.

How to Charge the AWS DeepRacer Drive Module Battery?

Follow the steps below to charge your AWS DeepRacer drive module battery:

1. Optionally remove from the vehicle the drive module battery.
2. Attach the battery charger to the battery, as depicted as follows:



3. Plug the power cord of battery charger into a power outlet.

How to Charge the AWS DeepRacer Compute Module Battery?

Follow the steps below to charge your AWS DeepRacer compute module battery:

1. Optionally remove the compute module battery from the vehicle.
2. Attach the compute power charger to the compute module battery.
3. Plug the power cord of the compute battery charger into a power outlet.

How to Maintain Vehicle's Wi-Fi Connection?

The following troubleshooting guide provides you tips for maintaining your vehicle's connection.

Topics

- [How to Troubleshoot Wi-Fi Connection if Vehicle's Wi-Fi LED Indicator Flashes Blue, Then Turns Red for Two Seconds, and Finally Off? \(p. 119\)](#)
- [What Does It Mean When the Vehicle's Wi-Fi or Power LED Indicator Flashes Blue? \(p. 119\)](#)
- [How Can I Connect to Vehicle's Device Console Using its Hostname? \(p. 120\)](#)
- [How to Connect to Vehicle's Device Console Using its IP Address? \(p. 120\)](#)

How to Troubleshoot Wi-Fi Connection if Vehicle's Wi-Fi LED Indicator Flashes Blue, Then Turns Red for Two Seconds, and Finally Off?

Check the following to verify you have the valid Wi-Fi connection settings.

- Verify that the USB drive has only one disk partition with only one *wifi-creds.txt* file on it. If multiple *wifi-creds.txt* files are found, all of them will be processed in the order they were found, which may lead to unpredictable behavior.
- Verify the Wi-Fi network's SSID and password are correctly specified in *wifi-creds.txt* file. An example of this file is shown as follows:

```
#####
# AWS DeepRacer #
# File name: wifi-creds.txt #
# #
# ...
#####
#
# Provide your SSID and password below
ssid: 'MyHomeWi-Fi'
password: myWiFiPassword
```

- Verify both the field names of *ssid* and *password* in the *wifi-creds.txt* file are in lower case.
- Verify that each of the field name and value is separated by one colon (:). For example. *ssid : 'MyHomeWi-Fi'*
- Verify that the field value containing a space is enclosed by a pair of single quotes. On Mac,TextEdit or some other text editor shows single quotes as of the '...' form, but not of '...'. If the field value does not contain spaces, the value can be without single quotes.

What Does It Mean When the Vehicle's Wi-Fi or Power LED Indicator Flashes Blue?

If the USB drive contains *wifi-creds.txt* file, the Wi-Fi LED indicator flashes blue while the vehicle is attempting to connect to the Wi-Fi network specified in the file.

If the USB drive has the *models* directory, the Power LED flashes blue while the vehicle is attempting to load the model files inside the directory.

If the USB drive has both the `wifi-creds.txt` file and the `models` directory, the vehicle will process the two sequentially, starting with an attempt to connect to Wi-Fi and then loading models.

The Wi-Fi LED might also turn red for two seconds if the Wi-Fi connection attempt fails.

How Can I Connect to Vehicle's Device Console Using its Hostname?

When connecting to the vehicle's device console using its hostname, make sure you type:
`https://hostname.local` in the browser, where `hostname` value (of the `AMSS-1234` format) is printed on the bottom of the AWS DeepRacer vehicle.)

How to Connect to Vehicle's Device Console Using its IP Address?

To connect to the device console using IP address as shown in the `device-status.txt` file (found on the USB drive), make sure the following conditions are met.

- Check your laptop or mobile devices are in the same network as the AWS DeepRacer vehicle.
- Check if you have connected to any VPN, if so, disconnect first.
- Try a different Wi-Fi network. For example, turn on personal hotspot on your phone.

How to Get the MAC Address of Your AWS DeepRacer Device?

Follow the instructions below to get the MAC address of your AWS DeepRacer device:

1. Make sure that your AWS DeepRacer device is only connected to a Wi-Fi network.
2. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
3. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.
4. Type `deepracer` in the **Username** input field.
5. Type the device SSH password in the **Password** input field.

If this is your first time to log in to the device, type `deepracer` in the **Password** input field. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.

6. After logged in, open a Terminal window.

You can use the Search button for the Terminal application.

7. Type the following Ubuntu shell command in the Terminal window:

```
ifconfig | grep HWaddr
```

The command produces an output similar to the following:

```
mlan0      Link encap:Ethernet    HWaddr 01:2a:34:b5:c6:de
```

The hexadecimal numbers are the device's MAC address.

How to Recover Your AWS DeepRacer Device Console Default Password?

Recovering your AWS DeepRacer device console default password involves retrieving or resetting the default password.

You can retrieve the default password by looking for the **Password** value printed at the bottom of the device.



Alternatively, you can log in to your AWS DeepRacer device to retrieve and to reset the password in a Ubuntu shell.

Follow the instructions below to recover the password of your AWS DeepRacer device web server using a Ubuntu Terminal window.

1. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
2. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.
3. Type `deepracer` in the **Username** input field.
4. Type the device SSH password in the **Password** input field.

If this is your first time to log in to the device, type `deepracer` in the **Password** input field. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.

5. After logged in, open a Terminal window.

You can use the Search button for the Terminal application.

6. To get the default device console password, type the following command in the Terminal window:

```
$cat /sys/class/dmi/id/chassis_asset_tag
```

The command outputs the default password as its result.

7. To reset the device console password to the default, run the following Python script from the Terminal window:

```
sudo python /opt/aws/deepracer/nginx/reset_default_password.py
```

How to Manually Update Your AWS DeepRacer Device?

Recent changes in the AWS DeepRacer service has made certain legacy devices, such as those distributed at AWS re:Invent 2018, unable to update automatically. Follow the steps below to manually update such a device.

To manually update an AWS DeepRacer device:

1. Download to your computer and unzip this [manually update a AWS DeepRacer device script](#).

The default name of the uncompressed file for this script is `deepracer-device-manual-update.sh`. In this topic, we'll assume you use this default script file name.

2. Copy the downloaded and uncompressed the script file (`deepracer-device-manual-update.sh`) from your computer to a USB drive.
3. Connect the device to a monitor using a HDMI-HDMI cable, to a USB keyboard, and to a USB mouse.
4. Power on the device and sign into the OS after the device is booted up.

You'll need to set the new OS password, if this is your first sign-in to the device.

5. Plug in the USB drive into the device and copy the script file to a folder (e.g., `~/Desktop`) on the device.
6. From a terminal on the device, type the following command to go to the script file's folder and to add execution permission to the script file:

```
cd ~/Desktop  
chmod +x deepracer-device-manual-update.sh
```

7. Type the following shell command to run the script:

```
sudo ./deepracer-device-manual-update.sh
```

8. When done with updating the device, open a web browser on your computer or a mobile device and navigate to the device IP address, e.g., 192.168.1.11 in a home network or 10.56.101.13 in an office network.

Make sure that your device is connected to your Wi-Fi network and use a browser in the same network without tunneling through a VPN.

9. On the device console, type the password for the device console to sign in. Wait for the update screen to show up and then follow the instructions therein.

How to Restore Your AWS DeepRacer Vehicle to Factory Settings?

At some point of time after trying out your AWS DeepRacer vehicle, you may want to reset the device to its factory settings and start afresh. The discussions presented below explains how to restore your AWS DeepRacer vehicle to factory settings.

Topics

- [Prepare for Factory Reset to Your AWS DeepRacer Vehicle \(p. 123\)](#)
- [Restore Your AWS DeepRacer Vehicle to Factory Settings \(p. 136\)](#)

Prepare for Factory Reset to Your AWS DeepRacer Vehicle

To prepare for the factory reset, perform the following tasks.

- Format the USB drive into the following two partitions.
 - FAT32 of 2 GB
 - NTFS of at least 16 GB
- Make the USB drive bootable to start the factory reset on reboot.
 - Burn the required custom Ubuntu ISO image to the FAT32 partition.
 - Copy the required factory restore files to the NTFS partition of the USB drive.

Depending on the computer you use, specific tasks may differ from one operating system to another. To illustrate, we present the step-by-step instructions to prepare the USB drive using a computer running Ubuntu (the computer module of the AWS DeepRacer vehicle) and also a computer running Windows.

The instructions for using other Linux or Unix computers are similar to the Ubuntu instructions discussed below. Just replace the apt-get commands with the corresponding commands supported by the other Linux or Unix system of your choosing.

Before starting, have the following items ready:

- One USB flash drive of at least 32 GB capacity.

- Your AWS DeepRacer vehicle to restore the factory settings to.
- A computer, when not using your AWS DeepRacer vehicle's compute module, to partition the USB drive and to make it bootable.

Choose one of the procedures below according to the type of your computer.

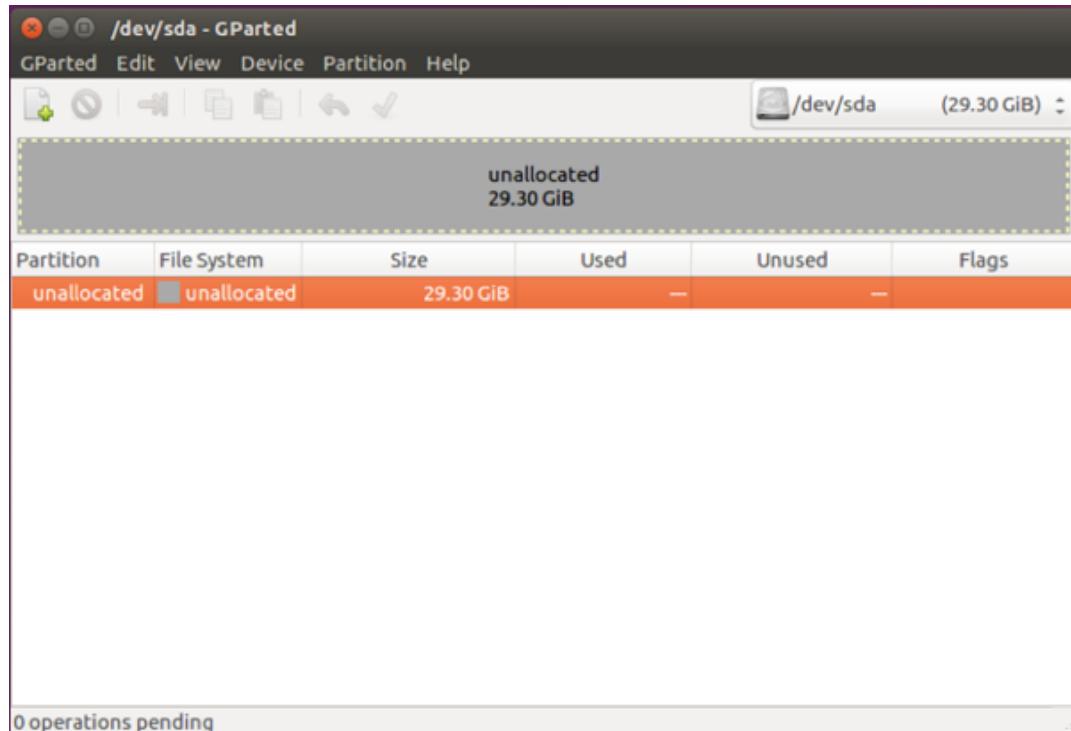
Partition a USB Drive and Make it Bootable by Using an Ubuntu Computer

In the following steps, use the AWS DeepRacer vehicle computer module as an Ubuntu computer. The same instructions apply to a Linux computer running Ubuntu. The instructions on other flavors of Linux or Unix operating systems are similar. Just replace the `apt-get *` commands with their corresponding commands supported by the other Linux or Unix system of your choosing.

To partition the USB drive and make it bootable

1. To format the USB drive by running Ubuntu commands on the AWS DeepRacer vehicle or a computer running Ubuntu, do the following.
 - a. On the AWS DeepRacer vehicle compute module, run the following commands to install and launch GParted.

```
sudo apt-get update; sudo apt-get install gparted  
sudo gparted
```



- b. On the newly created GParted console, choose `/dev/sda` in the drop-down menu on the top-right corner and then delete all existing partitions.

If the partitions are locked, open the context (right-click) menu and choose **unmount**.

- c. To create the FAT32 partition of 2 GB capacity, choose the file icon on the top-left, set the parameters similar to the following, and then choose **Add**.

Free space preceding: 1

New size:2048

Free space following: 27951

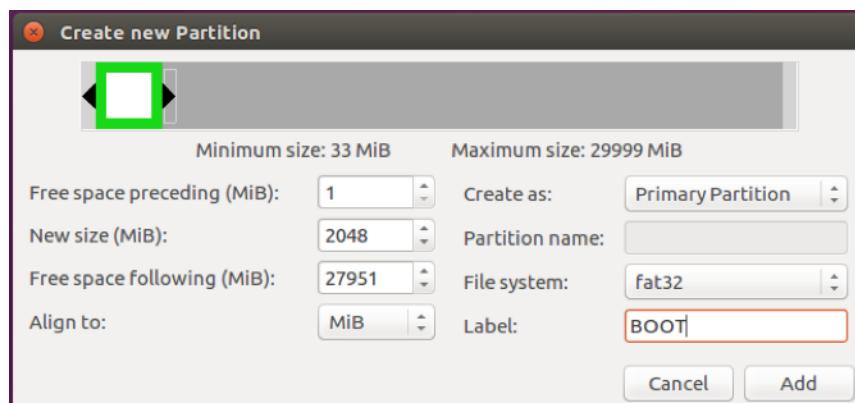
Align to: MiB

Create as: Primary Partition

Partition name:

File system: fat32

Label: BOOT



- d. To create the NTFS partition of at least 16 GB capacity, choose the file icon again, set the parameters similar to the following, and choose **Add**.

Free space preceding: 0

New size:27951

Free space following: 0

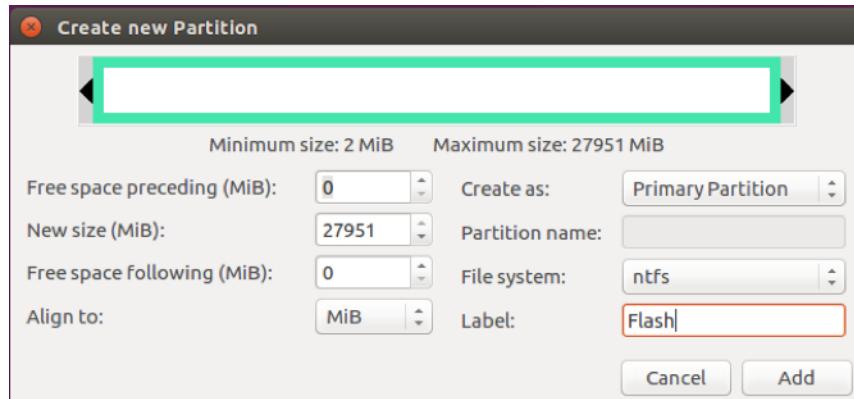
Align to: MiB

Create as: Primary Partition

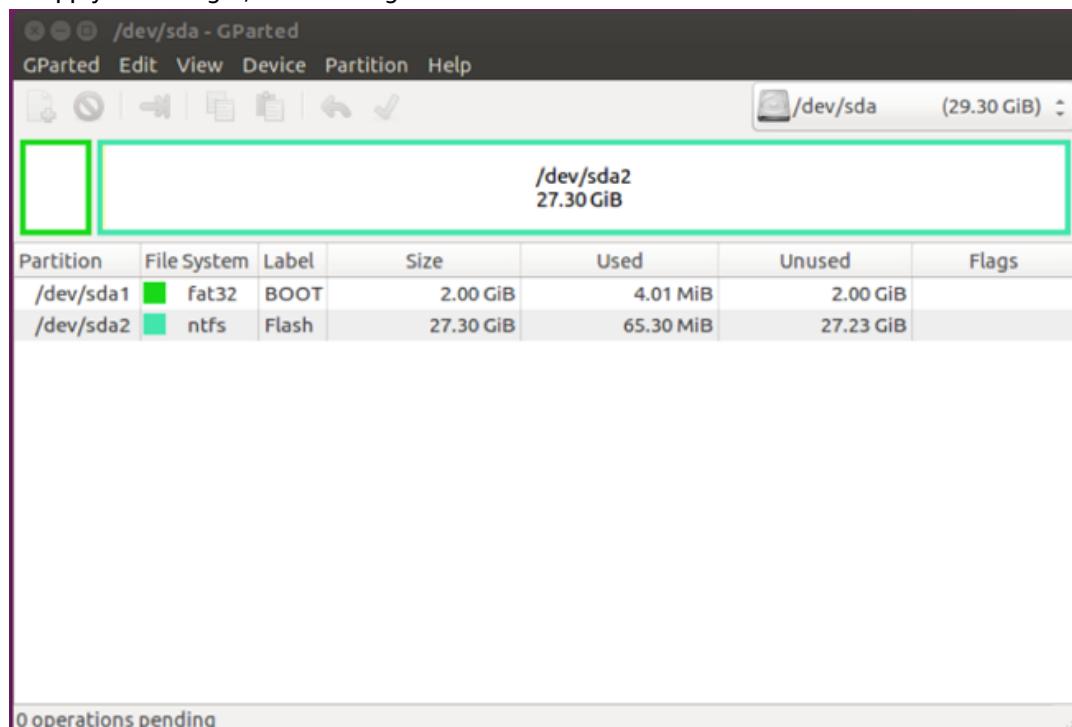
Partition name:

File system: ntfs

Label: Flash



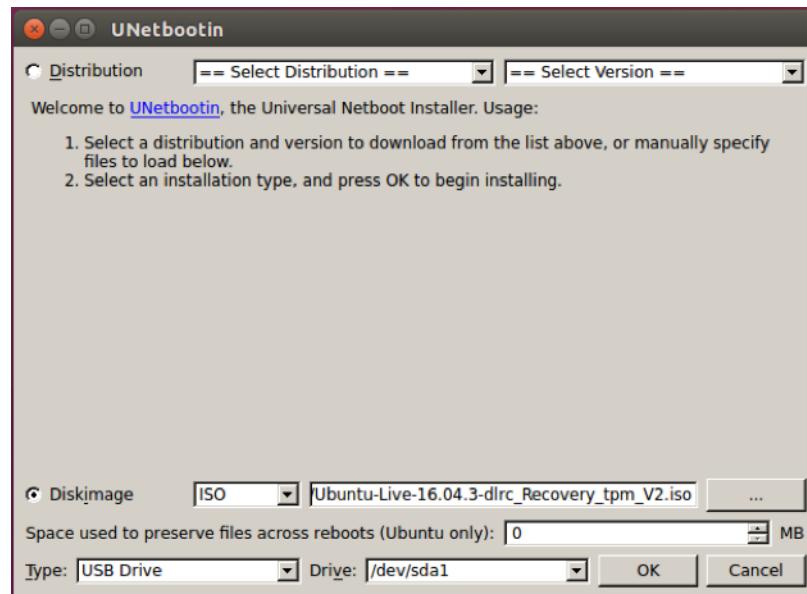
- e. To apply the changes, choose the green tick.



2. To make the USB drive bootable from the FAT32 partition, follow these steps.
 - a. Download the [customized Ubuntu ISO image](#).
 - b. Use UNetbootin on your AWS DeepRacer device, to do the following:
 - i. On your AWS DeepRacer compute module, run the following command to install and launch UNetbootin.

```
sudo apt-get update; sudo apt-get install unetbootin
sudo unetbootin
```
 - ii. On the UNetbootin window, do the following:
 - A. Check the **Disimage** radio button.
 - B. For the disk image, choose **ISO** from the drop-down menu.
 - C. Open the file picker to choose the downloaded Ubuntu ISO file.

- D. For **Type**, choose **USB Drive**.
- E. For **Drive**, choose `/dev/sda1`.
- F. Choose **OK**.



Note

The customized Ubuntu image may be more recent than what's shown here. If so, use the most recent version of the Ubuntu image.

If you get a **/dev/sda1 not mounted** alert message, choose OK to close the message, unplug the USB drive, replug the drive, and then follow the steps above to create the Ubuntu ISO image.

3. To copy the factory restore files to the NTFS partition of the USB drive, follow these steps.

- a. [Download](#) the compressed factory restore package. It's about 3.5 GB in size.
- b. Unzip the downloaded package, and
- c. Copy the following uncompressed files to the second (NTFS) partition of the USB drive:
 - Image files. About 9 GB:
 - `image_dlrc_1109_18WW45.5-2.bin`
 - `image_dlrc_1109_18WW45.5-2.bin.md5`
 - Script files:
 - `usb_flash.sh`
 - `set_hostname.py`
 - `dlrc_key.py`

Partition a USB Drive and Make it Bootable by Using a MacOS Computer

Follow the instructions here to use a MacOS computer to prepare the USB drive for factory reset.

To partition the USB drive and make it bootable by using a MacOS computer

1. To format the USB drive, follow these steps.

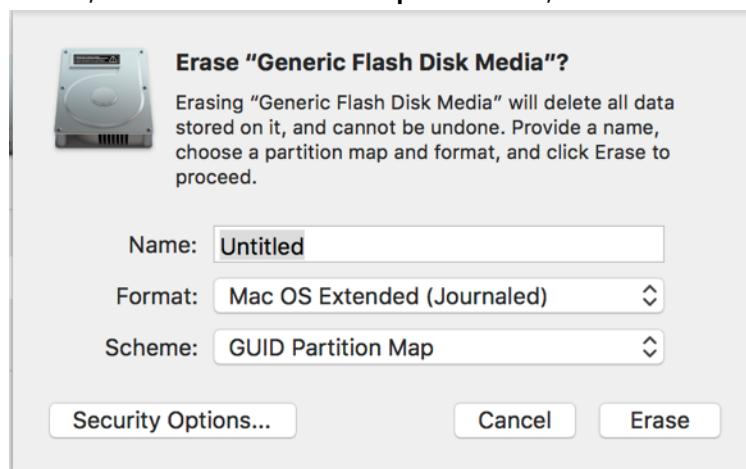
- a. Plug in the USB drive to your MacOS computer.
- b. Press command + space to open the search tool bar and then type Disk Utility.

Alternatively, you can choose **Finder->Applications->Utilities->Disk Utility** to open the Disk Utility.

- c. Choose the plugged-in USB drive, e.g., **Generic Flash Disk** which may differ from your USB drive name, on the left pane of Disk Utility. Then choose **Erase** on the top.



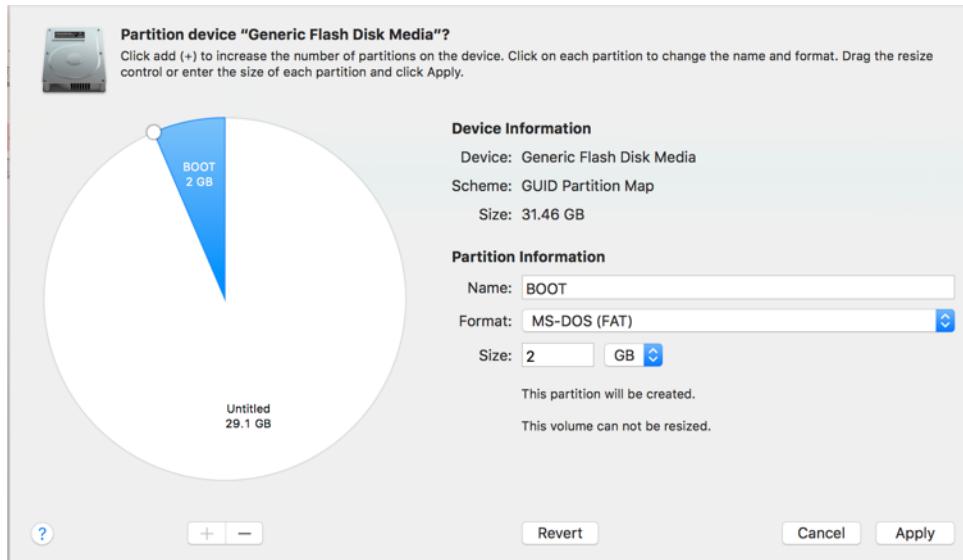
- d. On the **Erase "Generic Flash Disk Media"?** page, choose **Mac OS Extended (Journaled)** for Format, choose **GUID Partition Map** for Scheme, and then choose **Erase**.



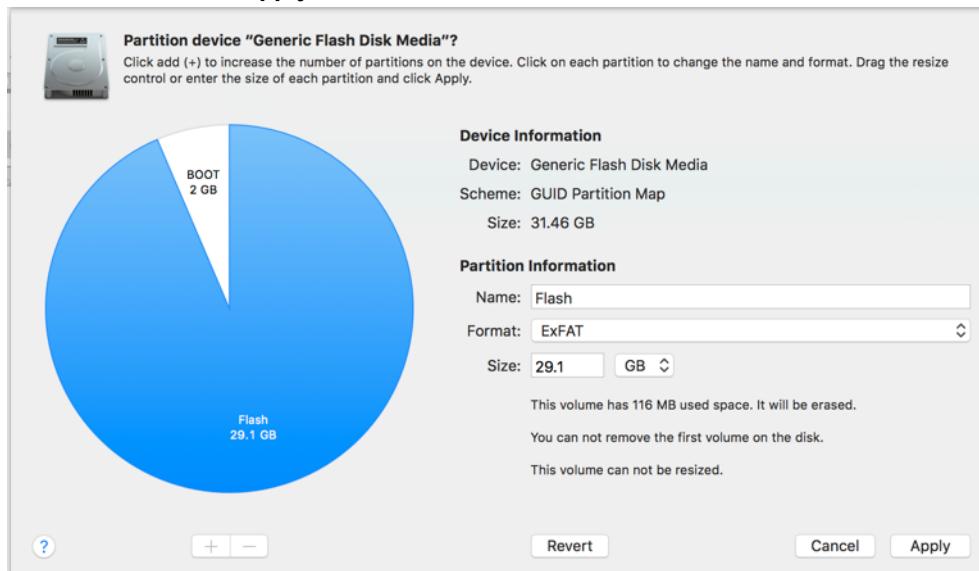
After the USB drive is refreshed, choose **Done** to continue on the **Erasing "Generic Flash Disk Media"?** dialog window.

- e. On the Disk Utility console, choose the USB drive on the left navigation pane, choose **Partition** from the menu on the top, and then choose the **+** button on the **Partition device ...** pop-up.

- f. To create the FAT32 partition of 2 GB capacity, under **Partition Information** type **Boot** (or another name of your choosing) for **Name**, choose **MS-DOS (FAT)** for **Format**, set **Size** to **2 GB**. Do not choose **Apply** yet.



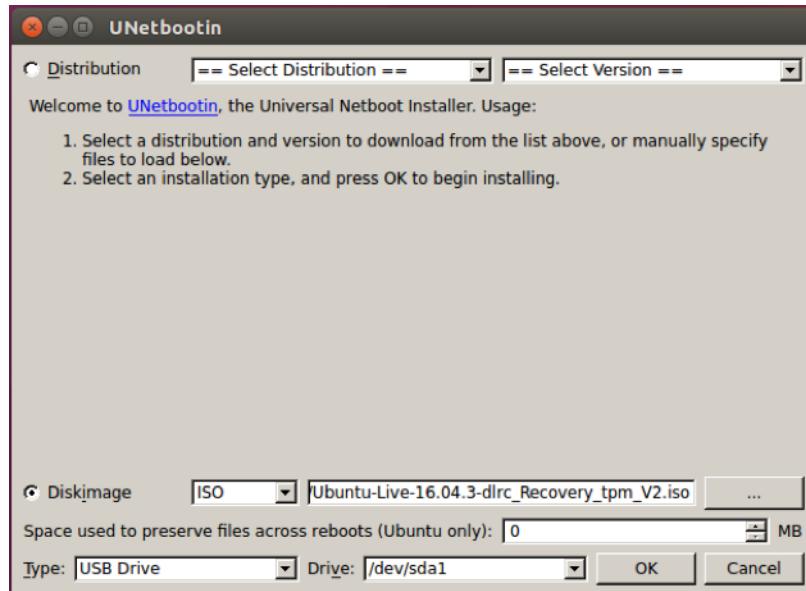
- g. To create the partition for the updated AWS DeepRacer image, choose a point in the other (**Untitled**) partition. Under **Partition Information**, type **Flash** (or another name of your choosing) for **Name**, choose **ExFat** for **Format**, leave the remaining capacity (in GB) of the USB drive in **Size**. Choose **Apply**.



- h. On the ensuing pop-up window, choose **Partition** to confirm creation of the specified new partitions.



- i. On the Disk Utility console, choose the **BOOT** partition on the left pane, and then choose **Info** from the menu on the top. Make note of the **BSD device node** value. In this tutorial, the value is `dsa1`. You need to supply this path when making the USB drive bootable from the FAT32 partition.
2. To make the USB drive bootable from the FAT32 partition, follow these steps.
 - a. Download the [customized Ubuntu image](#).
 - b. Go to <https://unetbootin.github.io/> to download the *UNetbootin* software. Then start the *UNetbootin* console.
 - c. On the **UNetbootin** console, do the following:
 - i. Check the **Disimage** radio button.
 - ii. For the disk image, choose **ISO** from the drop-down menu.
 - iii. Open the file picker to choose the downloaded Ubuntu ISO file.
 - iv. For **Type**, choose **USB Drive**.
 - v. For **Drive**, choose `/dev/sda1`.
 - vi. Choose **OK**.



Note

The customized Ubuntu image may be more recent than what's shown here. If so, use the most recent version of the Ubuntu image.

If you get a **/dev/sda1 not mounted** alert message, choose OK to close the message, unplug the USB drive, replug the drive, and then follow the steps above create the Ubuntu ISO image.

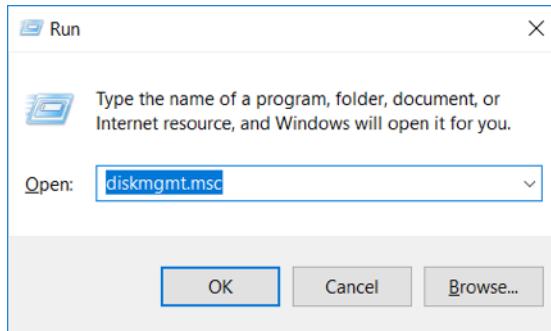
3. To copy the factory restore files to the NTFS partition of the USB drive, follow these steps.
 - a. [Download](#) the compressed factory restore package. It's about 3.5 GB in size.
 - b. Unzip the downloaded package.
 - c. Copy the following uncompressed files to the second (NTFS) partition of the USB drive:
 - Image files. About 9 GB:
 - *image_dlrc_1109_18WW45.5-2.bin*
 - *image_dlrc_1109_18WW45.5-2.bin.md5*
 - Script files:
 - *usb_flash.sh*
 - *set_hostname.py*
 - *dlrc_key.py*

Partition a USB Drive and Make it Bootable by Using a Windows Computer

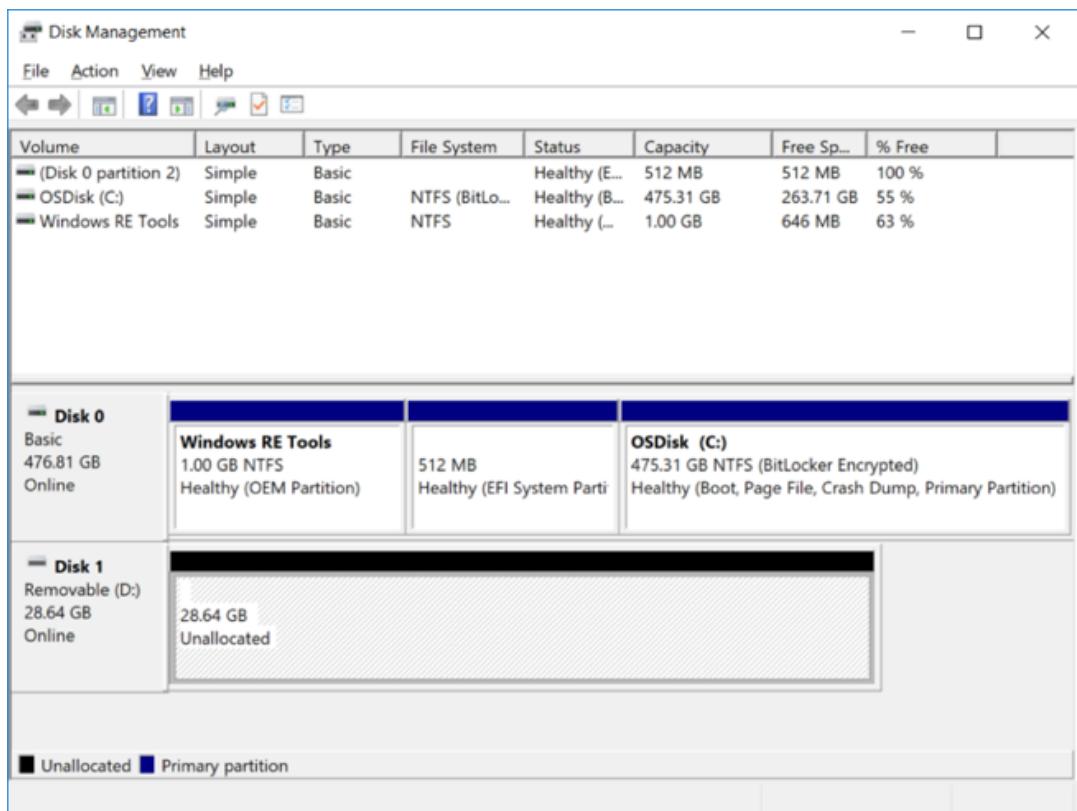
Follow the instructions here to use a Windows computer to prepare the USB drive for factory reset.

To partition the USB drive and make it bootable by using a Windows computer

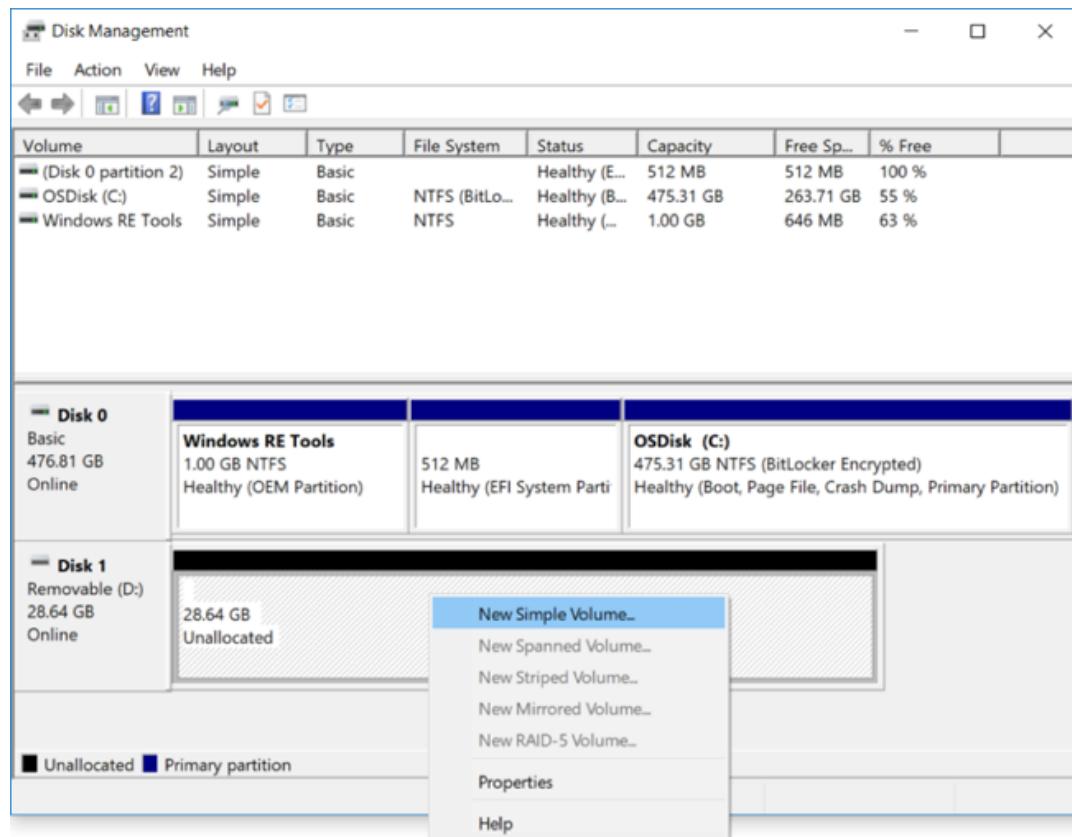
1. To format the USB drive, follow these steps.
 - a. Open the Windows command prompt, type `diskmgmt.msc`, and choose **OK** to launch the Windows Disk Management Console.



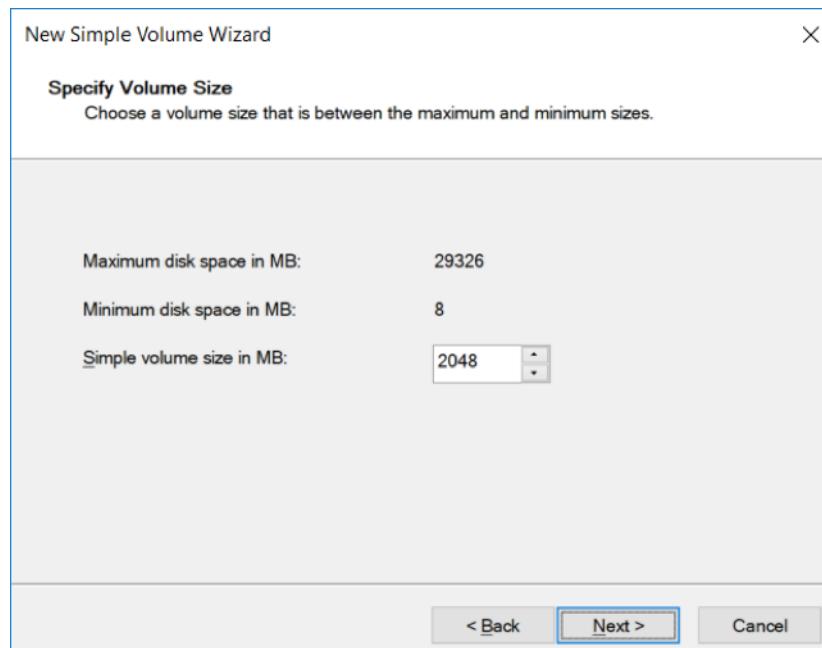
- b. From the Disk Management console, choose the USB drive. Delete all the partitions and make the drive unallocated. The example in the screenshot here shows **Disk 1 Removable (D:)** as the USB drive.



- c. To create the FAT32 partition of 2 GB capacity, open the Disk Management console and choose the USB drive. Open the context (right-click) menu and choose **New Simple Volume** from the context menu.

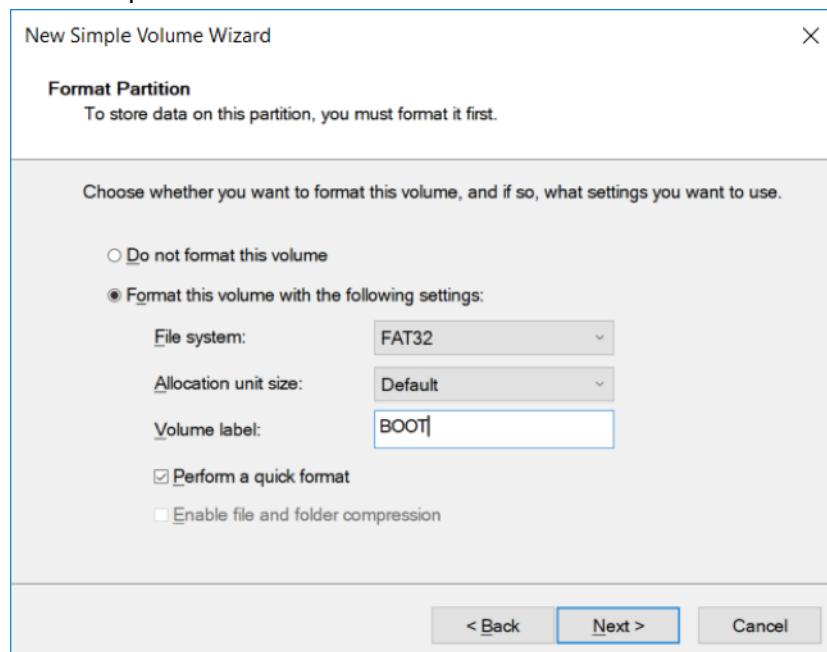


- d. On the New Simple Volume Wizard, choose **2048** for **Simple volume size in MB** and then choose **Next**.

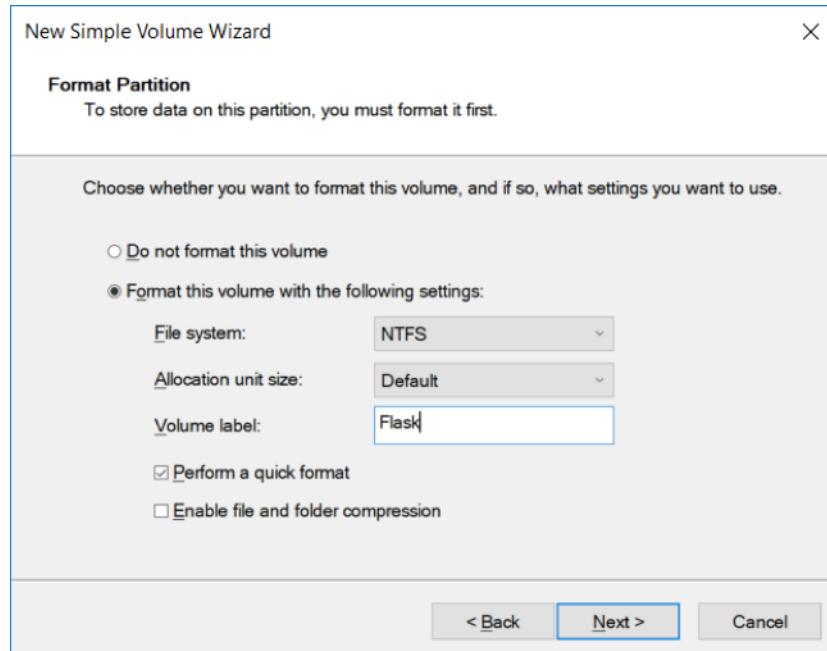


- e. On the **New Simple Volume Wizard** page and under **Format Partition**, choose the **Format this volume with the following settings**. Then, choose **FAT32** for **File system**, **Default** for

Allocation unit size and any label (e.g., `BOOT`) for **Volume label**. Finally, choose **Next** to create the FAT32 partition.

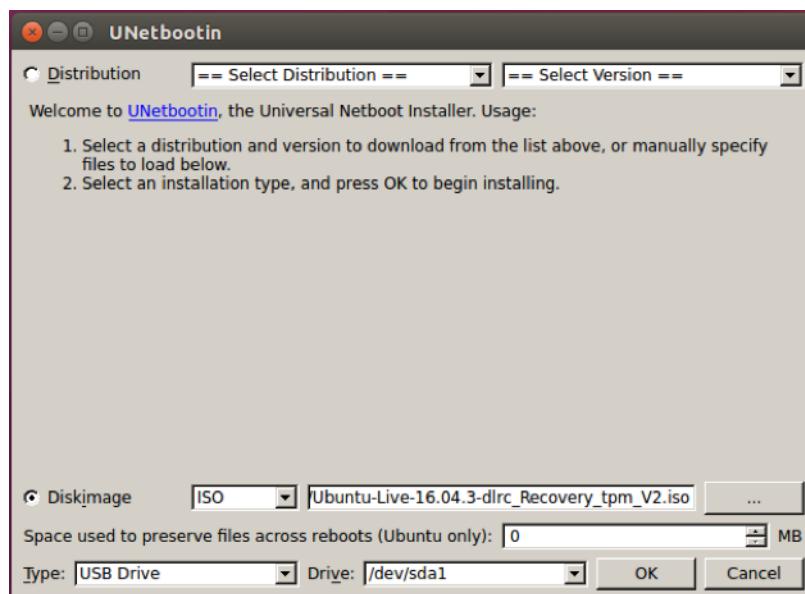


- f. To create the NTFS partition of the remaining disk capacity, open the Disk Management console. Choose the USB drive and open the context (right-click) menu to choose **New Simple Volume**. Choose the **Format this volume with the following settings** option. Choose NTFS for **File system**, Default for **Allocation unit size**, and a label (e.g, `Flask`) for **Volume label**. Finally, choose **Next** to start creating the NTFS partition.



2. To make the USB drive bootable from the FAT32 partition, follow these steps.
 - a. Download the [customized Ubuntu image](#).

- b. Go to <https://unetbootin.github.io/> to download the UNetbootin software. Then start the UNetbootin console.
- c. On the **UNetbootin** console, do the following:
 - i. Check the **Disimage** radio button.
 - ii. For the disk image, choose **ISO** from the drop-down menu.
 - iii. Open the file picker to choose the downloaded Ubuntu ISO file.
 - iv. For **Type**, choose **USB Drive**.
 - v. For **Drive**, choose `/dev/sda1`.
 - vi. Choose **OK**.



Note

The customized Ubuntu image may be more recent than what's shown here. If so, use the most recent version of the Ubuntu image.

If you get a **/dev/sda1 not mounted** alert message, choose OK to close the message, unplug the USB drive, replug the drive, and then follow the steps above to create the Ubuntu ISO image.

3. To copy the factory restore files to the NTFS partition of the USB drive, follow these steps.
 - a. **Download** the compressed factory restore package. It's about 3.5 GB in size.
 - b. Unzip the downloaded package. If your favorite tool can't unzip the file successfully, try using the PowerShell [Expand-Archive](#) command.
 - c. Copy the following uncompressed files to the second (NTFS) partition of the USB drive:
 - Image files. About 9 GB:
 - `image_dlrc_1109_18WW45.5-2.bin`
 - `image_dlrc_1109_18WW45.5-2.bin.md5`
 - Script files:
 - `usb_flash.sh`
 - `set_hostname.py`
 - `dlrc_key.py`

Restore Your AWS DeepRacer Vehicle to Factory Settings

Follow the instructions here to restore your AWS DeepRacer vehicle to its factory settings. Make sure you have made proper preparations as described in [the section called “Prepare for Factory Reset” \(p. 123\)](#).

Note

After the factory reset, all data stored on your AWS DeepRacer vehicle will be erased.

To restore your AWS DeepRacer vehicle to its factory settings

1. Insert the prepared USB drive to your AWS DeepRacer compute module. Turn on the power and repeatedly press the *ESC* key to enter BIOS.
2. From the BIOS window, choose **Boot From File**, then **The option with USB in it**, then **EFI**, then **BOOT**, and finally **BOOTx64.EFI**.
3. After the compute module is booted, wait for the device reset to start automatically when the power LED indicator starts to flash and a terminal window is presented to display the progress. You don't provide any further user input at this stage.

If some error happens and the recovery fails, restart the procedure from **Step 1**. For detailed error messages, see the *result.log* file generated on the USB drive.

4. Wait for about 6 minutes for the power LED to stop flashing when the terminal closes automatically and the factory reset completes. The device then reboots itself automatically.
5. After the device is restored to factory settings, disconnect the USB drive from the vehicle's compute module.

After the factory reset, your AWS DeepRacer vehicle software is likely outdated. To update the vehicle software, go to the AWS DeepRacer device console and follow the instructions.

Document History for AWS DeepRacer Developer Guide

The following table describes the important changes to the documentation since the last release of AWS DeepRacer.

update-history-change	update-history-description	update-history-date
Updates for general availability (p. 137)	AWS DeepRacer now features more robust methods to train and evaluate deep learning models. The user interface is updated and explained. More options and precise data is available to build your own physical tracks. Troubleshooting information is available.	April 29, 2019
Initial release AWS DeepRacer Developer Guide (p. 137)	Initial release of the documentation to help the AWS DeepRacer user to learn reinforcement learning and explore its applications for autonomous racing, using the AWS DeepRacer console, the AWS DeepRacer simulator, and a AWS DeepRacer scale model vehicle.	November 28, 2018

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.