AWS DEEPRACER BLOG

Table of Contents

1 AWS DeepRacer

AWS DeepRacer is an integrated learning system (ILS) for users enabling them to learn and discover supportive “reinforcement learning” mechanism and to research and create a self-directed driving application. It is comprised of the following three components:

1.1 AWS DeepRacer Console

It is a GUI to interrelate with AWS DeepRacer facilities. User can use the console to train a “reinforcement learning model” and to gauge its performance in AWS DeepRacer emulator that is built on AWS RoboMaker. Using this console, user can also download the already trained prototype for deployment to AWS DeepRacer vehicle for self-directed driving in physical environment. AWS console provides backings to the following features:

- Build a training job to assist in a “reinforcement learning model” with particular rewards, environment, optimized algorithms, and hyper-parameters.
- Select the simulated track for training and estimate the model with Amazon SageMaker & AWS RoboMaker.
- Generate the copy of a trained model to improve training using hyper-parameters to enhance the model’s performance.
- Download the already trained model to deploy in AWS DeepRacer in such a way that it drives in physical context.
- Put forward the model to virtual race and examine its performance rank wise against other models in virtual leaderboard.

1.2 AWS DeepRacer Vehicle

It is Wi-Fi enabled real vehicle that can automatically drive on a physical track with the use of “reinforcement learning model”. User can control its vehicle manually or can deploy an automatically driven model for it.

To work in an autonomous mode, it runs an inference on computing module of the vehicle. This inference makes use of the images that are taken from the camera mounted on the vehicle at the front.

Vehicle Integrated softwares are downloaded using Wi-Fi connection. It permits user to access the vehicle console to work with the control from mobile or computer gadgets.

1.3 AWS DeepRacer League

It is a significant component of the AWS DeepRacer and is envisioned to stand-in “communal learning” and joint exploration by taking part in competitions. With this league, the user can compare its developing skills with other AWS DeepRacer developers in virtual or physical racing events. It not only benefits the user with rewards but also provide way to measure its “reinforcement learning model (RLM)”.

2 How AWS DeepRacer Works?

AWS DeepRacer practices “reinforcement learning method (RLM)” to empower self-directed driving to the AWS DeepRacer vehicle. It can be achieved by training and evaluating the RLM in the virtual climate with the simulated track. After training the user uploads its trained article on the AWS vehicle and set it for self-governing driving in physical context with an actual track.

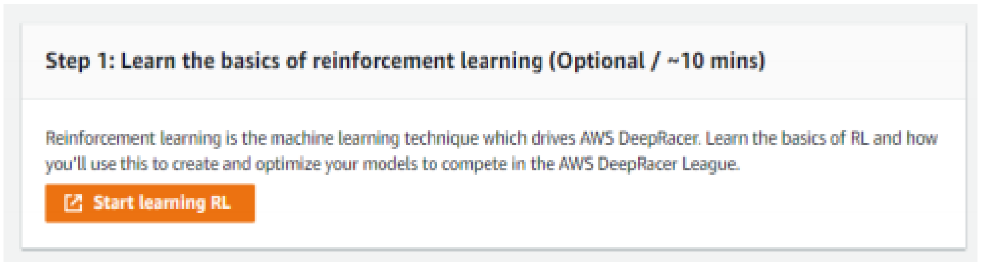
AWS DeepRacer vehicle is 1/18th scale vehicle that drives automatically on a track with race against other vehicles. This vehicle is well equipped with sensors such as stereo cameras, front-facing camera, LIDARs and radars etc. These sensors gather the data related to the environment where vehicle is to be operated.

3 How to Get Started with AWS DeepRacer?

Before starting with AWS DeepRacer, following are some of the steps to make use of AWS DeepRacer console to align the vehicle with the suitable sensors for self-governing automatic driving necessities, to train RLM for vehicle with particular sensors and to analyze these trained models for determining the excellence of trained model.

3.1 Training RL Model using AWS DeepRacer console

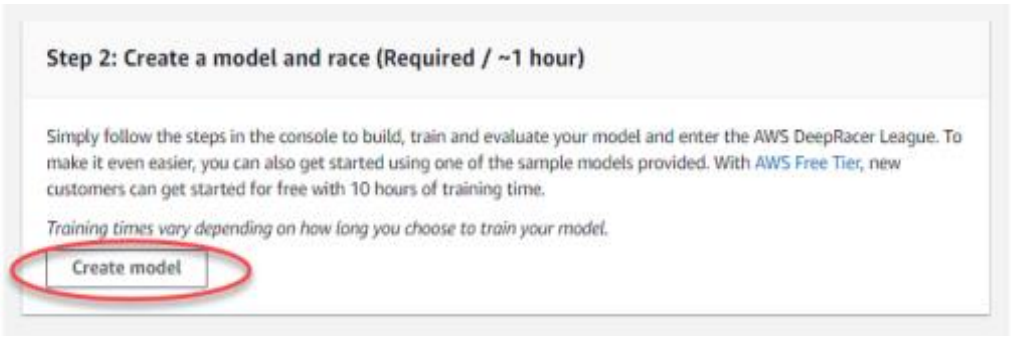
1. On the start, choose "Get started" from the service landing page or click on the "Get started with the reinforcement learning" from main navigation page.



Step 1: Learn the basics of reinforcement learning (Optional / ~10 mins)

Reinforcement learning is the machine learning technique which drives AWS DeepRacer. Learn the basics of RL and how you'll use this to create and optimize your models to compete in the AWS DeepRacer League.

2. After clicking on "Start learning RL", click on "create a model and race" and then click "Create Model".

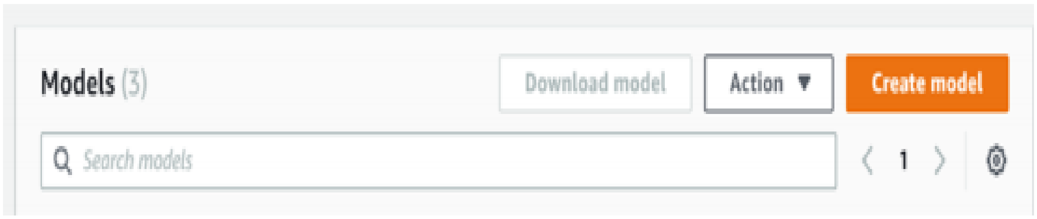


Step 2: Create a model and race (Required / ~1 hour)

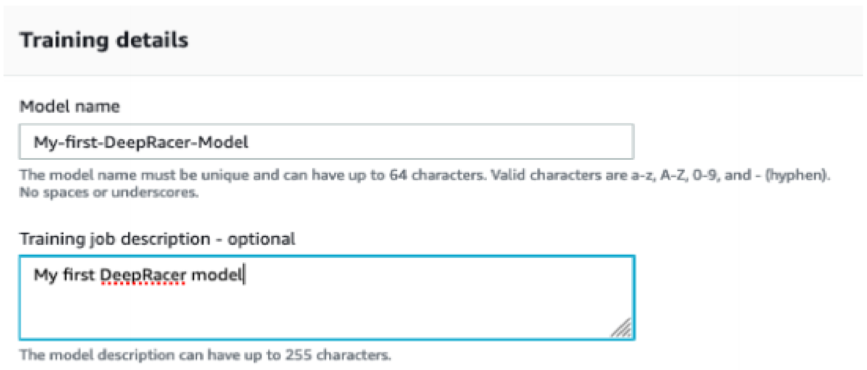
Simply follow the steps in the console to build, train and evaluate your model and enter the AWS DeepRacer League. To make it even easier, you can also get started using one of the sample models provided. With AWS Free Tier, new customers can get started for free with 10 hours of training time.

Training times vary depending on how long you choose to train your model.

Alternatively, user can choose "Your Models" option in AWS DeepRacer home page from navigation pane. On opening "Your Model" page, click "create model".



3. On the "Create model" page, under "Account resources", click on "Create resources". If user encounters any issue, click on "Reset Resources".
4. On "Create model" page, under "Training details", write name for model of user's choice in "Model name" tab. It also provides an option to write descriptive summary of user's model in "Training job description". This model name is used as reference model while submitting it to the leaderboard of AWS sponsored racing events or while cloning.


Training details

Model name

My-first-DeepRacer-Model

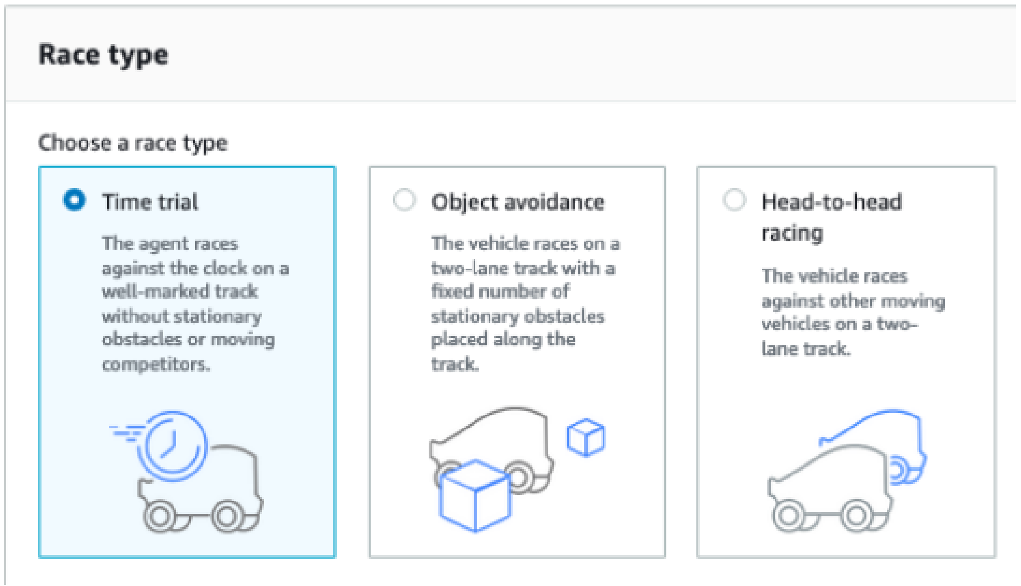
The model name must be unique and can have up to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces or underscores.

Training job description - optional

My first DeepRacer model

The model description can have up to 255 characters.

5. On "Create model" page, under "Environment simulation", select a track as "virtual environment" to train AWS DeepRacer vehicle and click Next. For first run, select track with simple outline and smooth turns. For later iterations, user can choose complex tracks to gradually improve the models. To train model for specific racing occasion, always select track that is most alike to event track.
6. On "Create model" page, click "Next".
7. On "Create Model" page, under "Race type", select "training type".


Race type

Choose a race type

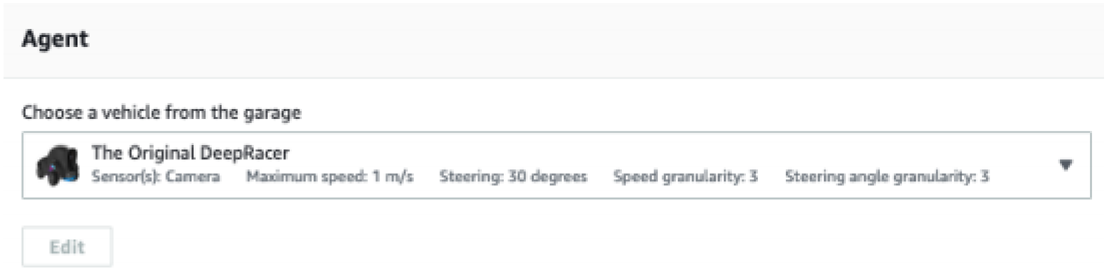
[✓ Time trial] The agent races against the clock on a well-marked track without stationary obstacles or moving competitors.

Object avoidance The vehicle races on a two-lane track with a fixed number of stationary obstacles placed along the track.

Head-to-head racing The vehicle races against other moving vehicles on a two-lane track.

It would be better if first time users select "time trial" as it deploys default sensor for configuration with single camera.

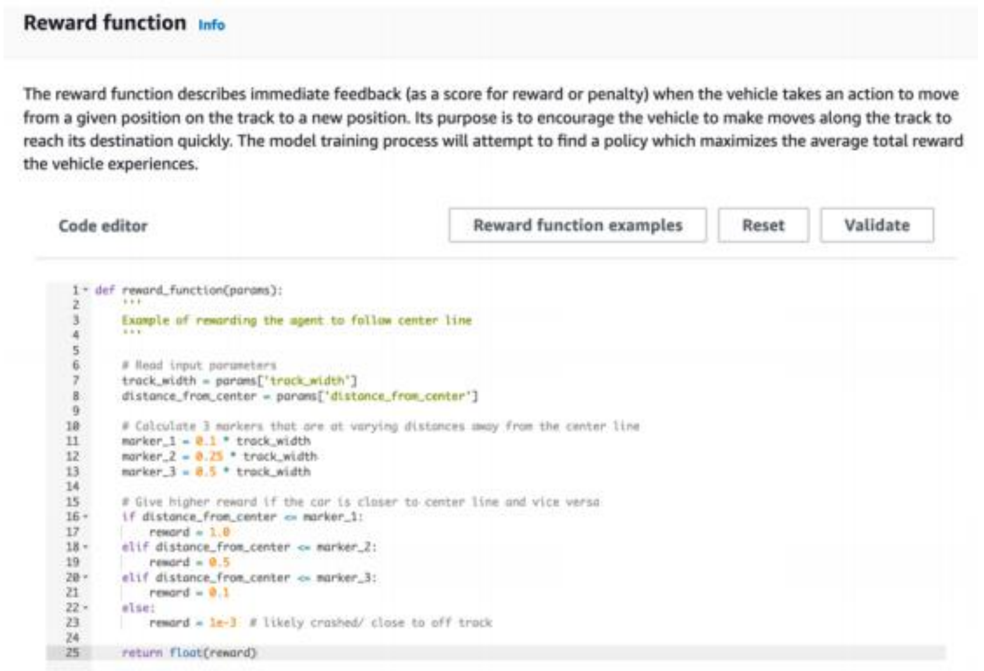
8. On "create model" page, under "Agent" select "The Original DeepRacer" while creating first model, and click "next".


Agent

Choose a vehicle from the garage

The Original DeepRacer
Sensor(s): Camera Maximum speed: 1 m/s Steering: 30 degrees Speed granularity: 3 Steering angle granularity: 3

9. On the same page, under "Reward function", consider "default reward function" for first model.



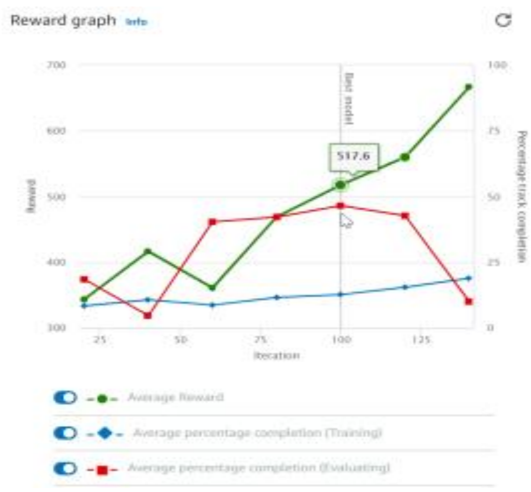
Reward function Info

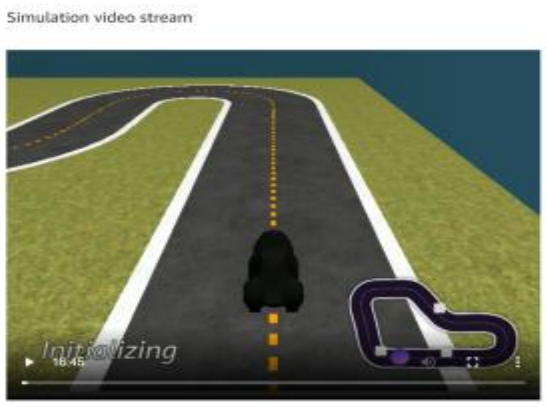
The reward function describes immediate feedback (as a score for reward or penalty) when the vehicle takes an action to move from a given position on the track to a new position. Its purpose is to encourage the vehicle to make moves along the track to reach its destination quickly. The model training process will attempt to find a policy which maximizes the average total reward the vehicle experiences.

Code editor

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

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

    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    return float(reward)
```

10. On “create model” page, under “training algorithm and hyper-parameters” select the default hyper-parameters. Once the model is created, user can expand and modify these hyper-parameters values to enhance its efficiency.
11. Under “stop conditions” options, keep the “default maximum time value” as it is. User can also select a new value for termination of training job. It will help preventing long-running training job.
    First time user should start with small value for this parameter and later on should gradually train for longer runs.
12. Now select “Create model” to start building model and provide the illustration for training job.
13. After submission, “initialization” of training job will begin and it will then run. It will take about 6 minutes from initializing the process to “in progress”.
14. To witness the progress of training job, see the “reward graph” and “simulation video stream”. “refresh” button can be clicked to refresh the reward graph until the training job is completed.





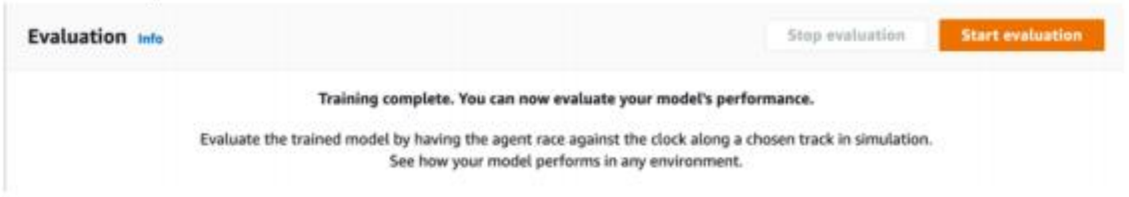
After the training job stops, the user can evaluate the trained model following the steps mentioned below.

3.2 Evaluating AWS DeepRacer Trained Model in Simulation

When the model training job is complete, user must evaluate and analyze the trained model to measure its convergence behavior. It is done by continuing number of trials on a selected track with the agent to move on the track according to the actions programmed by the trained model. Its performance metrices are "percentage of track completion" & "time taken while running from start to finish".

For evaluation of trained model, AWS DeepRacer console can be used keeping in view the following steps:

1. Open "AWS DeepRacer console" at (https://console.aws.amazon.com/deepracer).
2. From the main navigation pane, select "Models" and click on the user trained model from the model list.
3. In "Evaluation", click "Start evaluation".

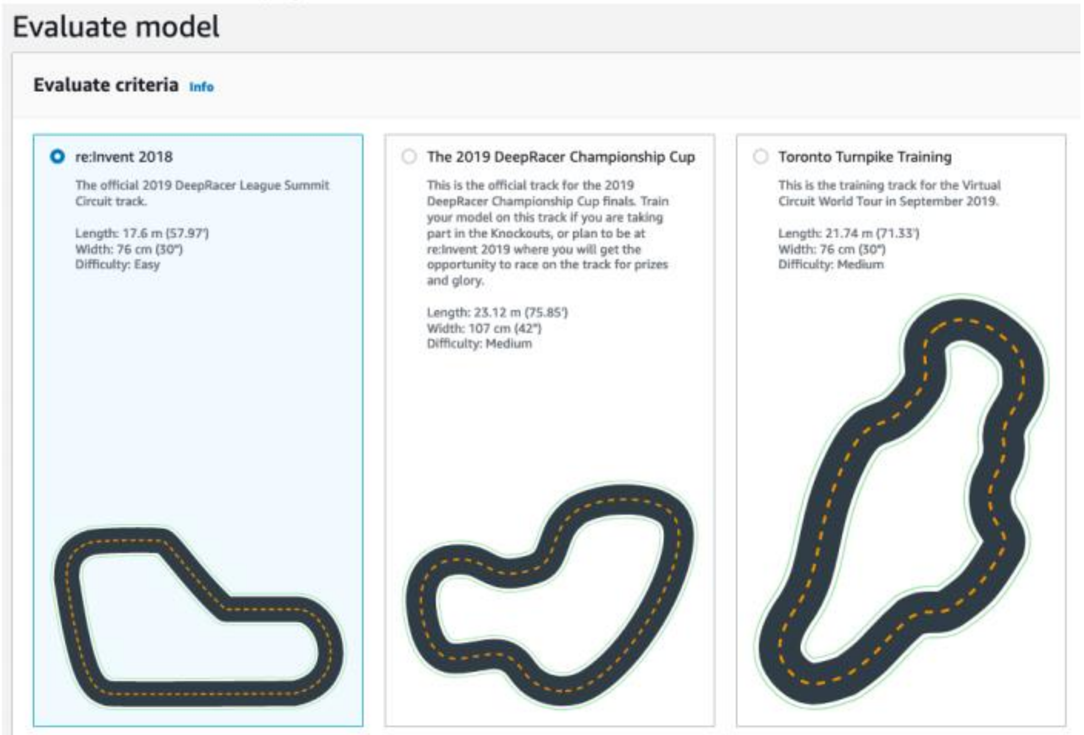


Training complete. You can now evaluate your model's performance.

Evaluate the trained model by having the agent race against the clock along a chosen track in simulation.
See how your model performs in any environment.

User can start evaluation if the training job status appears to be "completed" or else it would change to "ready", if not completed.

4. On the "Evaluate model" page, under "Evaluate criteria", select the "track under Evaluation criteria". It must be preferably same as used while training model.



re:Invent 2018
The official 2019 DeepRacer League Summit Circuit track.
Length: 17.6 m (57.97')
Width: 76 cm (30")
Difficulty: Easy

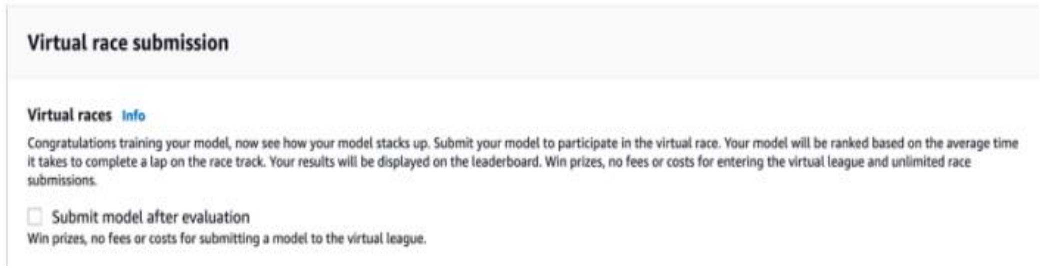
The 2019 DeepRacer Championship Cup
This is the official track for the 2019 DeepRacer Championship Cup finals. Train your model on this track if you are taking part in the Knockouts, or plan to be at re:Invent 2019 where you will get the opportunity to race on the track for prizes and glory.
Length: 23.12 m (75.85')
Width: 107 cm (42")
Difficulty: Medium

Toronto Turnpike Training
This is the training track for the Virtual Circuit World Tour in September 2019.
Length: 21.74 m (71.33')
Width: 76 cm (30")
Difficulty: Medium

5. On the same page, under "Evaluate Criteria" select "number of trials" that user wants to evaluate model.
6. Under "Race type", click on the "racing type" similar to that selected at the time of training the model.

For evaluation user can select a different "race type" other than selected while training. For instance, user can train the model for "head-to-head races" but can evaluate it for "time trials". Though, model must be generalized well but it is recommended to use same race type both for "evaluation" and "training".

7. Under "Virtual Race Submission", turn off "Submit model" after "evaluation option" for the first submitted model. It is enabled when model is to be participated in the racing event.



Virtual race submission

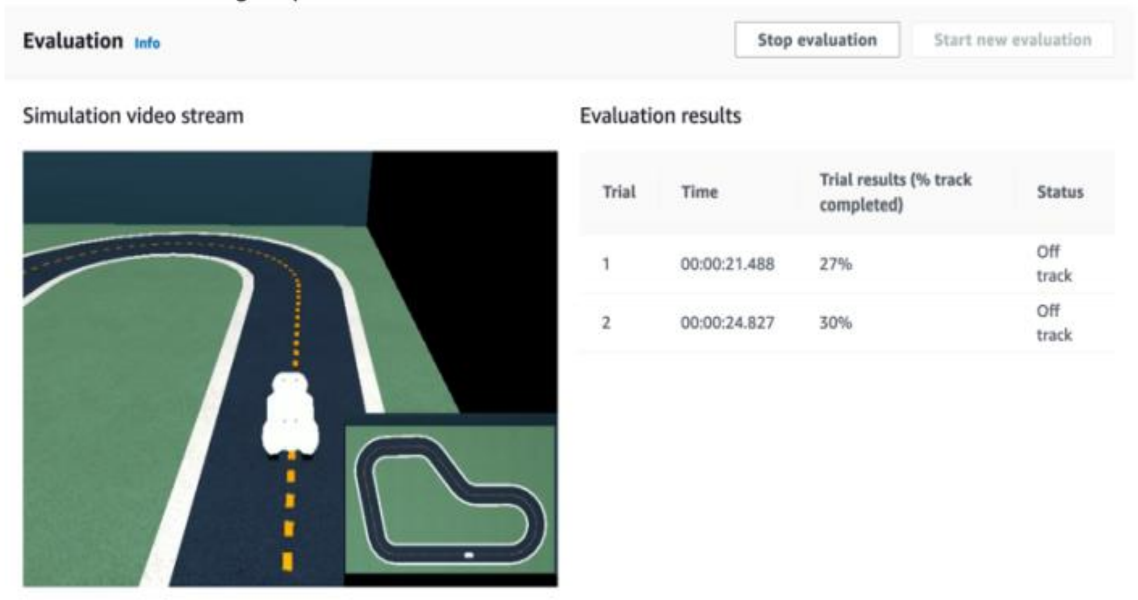
Virtual races Info

Congratulations training your model, now see how your model stacks up. 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. Your results will be displayed on the leaderboard. Win prizes, no fees or costs for entering the virtual league and unlimited race submissions.

☐ Submit model after evaluation

Win prizes, no fees or costs for submitting a model to the virtual league.

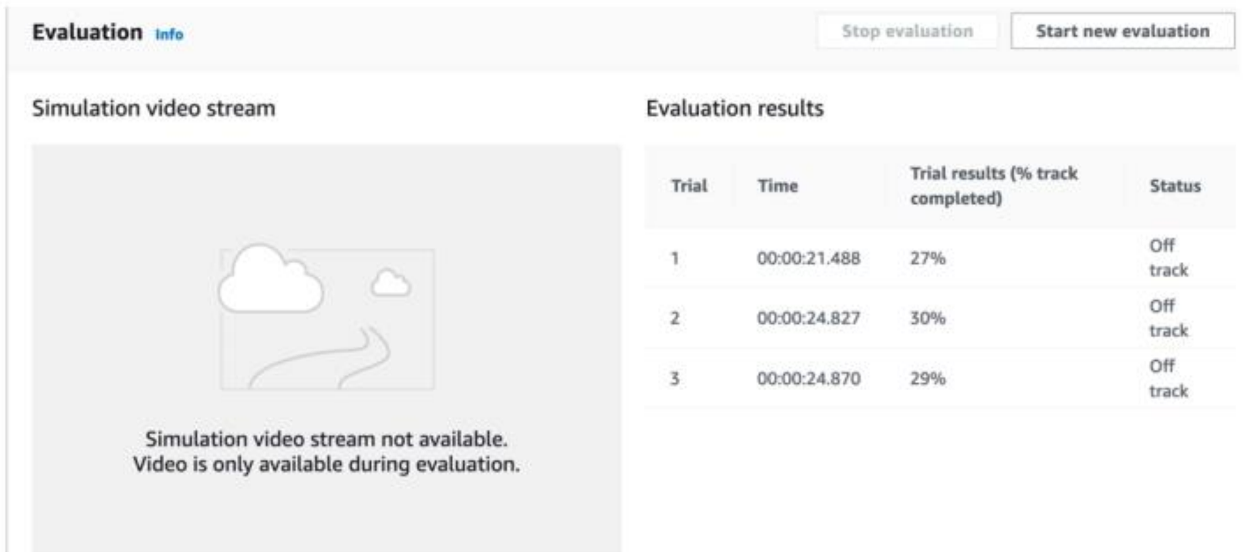
8. On the same page, select "start evaluation" to start creating and initializing of evaluation job. It will take about 3 minutes to complete.
9. As evaluation occurs, evaluation results such as "trial time" and "track completion rate" are presented under "Evaluation" after every trial. In the "Simulation video stream window", user can see and evaluate the performance vehicle (agent) on selected track. Evaluation can be stopped before track completion. Click on the "Stop evaluation" button at the upper right corner of the "Evaluation card" to stop the evaluation, it asks the confirmation to stop and then stops it.



Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:21.488	27%	Off track
2	00:00:24.827	30%	Off track

10. When the evaluation job is done, analyze the performance metrices of all trials under "Evaluation results". While analyzing, "simulation video stream" will no longer be available as it is there only during evaluation process.
11. In this particular training and evaluation job, model failed to complete the trials. It is because of so many possible reason. It is because training model did not converge. It needs more time. Agent required a large room or reward function needed upgradation to handle the changing environment.


Evaluation
Info
Stop evaluation
Start new evaluation
Simulation video stream
Simulation video stream not available.
Video is only available during evaluation.

Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:21.488	27%	Off track
2	00:00:24.827	30%	Off track
3	00:00:24.870	29%	Off track

4 Train and Evaluate the AWS DeepRacer Model using AWS DeepRacer Console

DeepRacer console is used to train the RL model of which step by step instructions has been elaborated in the previous section. In AWS console, at first training job is created, and supportive framework is selected with the algorithm available, then reward function is added and training settings are configured.

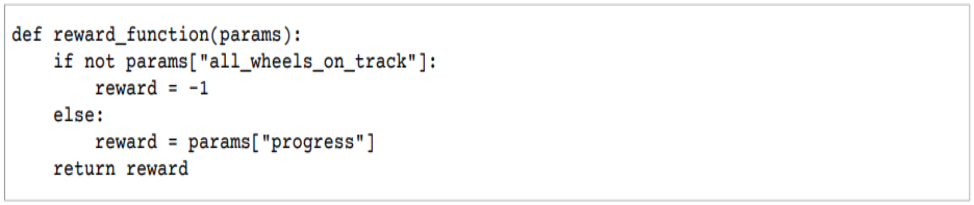
4.1 Creating Reward Function

Reward function is the immediate feedback for the vehicle when it moves from one point to the other on the track. This reward encourages the vehicle to make steps to move on the track to reach its destination without any interference. Positive reward receives when the vehicle moves along the desirable track in an appropriate manner. Reward function is like the incentive that user can provide to its vehicle. If it is not considered carefully, it can lead to the inadvertent consequences of contrary effect. Below are some reward function examples that can be taken into consideration while training the RL model using AWS console.

1. First example is to reward the AWS vehicle for moving on a ***straight path*** from starting point to the finishing point. It depends on the two function logics i.e., “on track” & “progress”.

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

   This logic fines the agent when it moves off the track and give rewards when it comes to the finishing line. Though rewarding this way is achievable but here agent can move freely on the track and still would get reward when it reaches at the end even after taking so long time. It makes the model less efficient, so it would be better if the vehicle is rewarded bit-by-bit throughout the training path. For agent to drive on straight track, reward function can be improved the reward as follows:

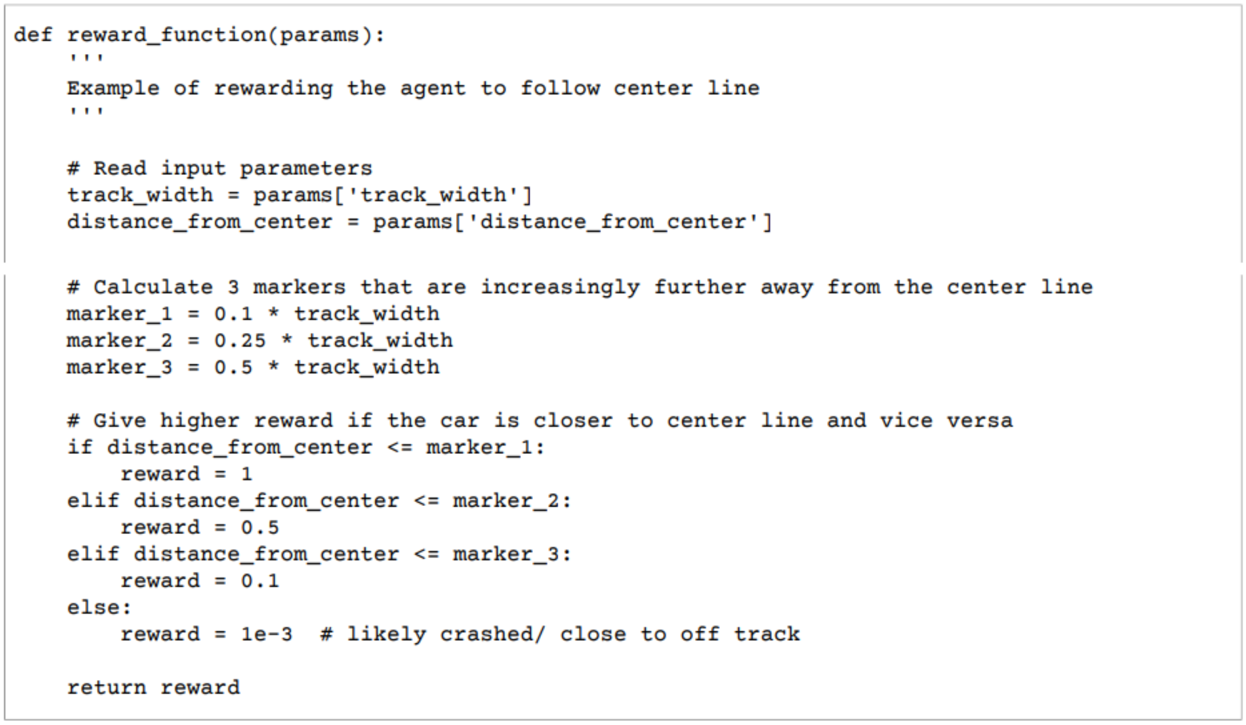


```
def reward_function(params):
    if not params["all_wheels_on_track"]:
        reward = -1
    else:
        reward = params["progress"]
    return reward
```

Using this function, the vehicle gets more reward when closer it reaches to the finishing line. Moreover, it reduces the unproductive trials of the vehicle to drive backward. It will not only increase the efficiency of eth vehicle but also it would reach the destination on time when implemented in real time scenario.

User should enhance the reward when agent moves on a path other than straight track because in looped paths, vehicle cannot stay on the track as reward function ignores the actions to make the turn to follow the curved paths. Some of the examples regarding enhanced reward function is as follows:

2. **Example to follow center line in time trials:** This example shows the distance of agent from the central line on the track and rewards high if the agent is moving closer to the center line.



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

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

    # Calculate 3 markers that are increasingly further 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
```

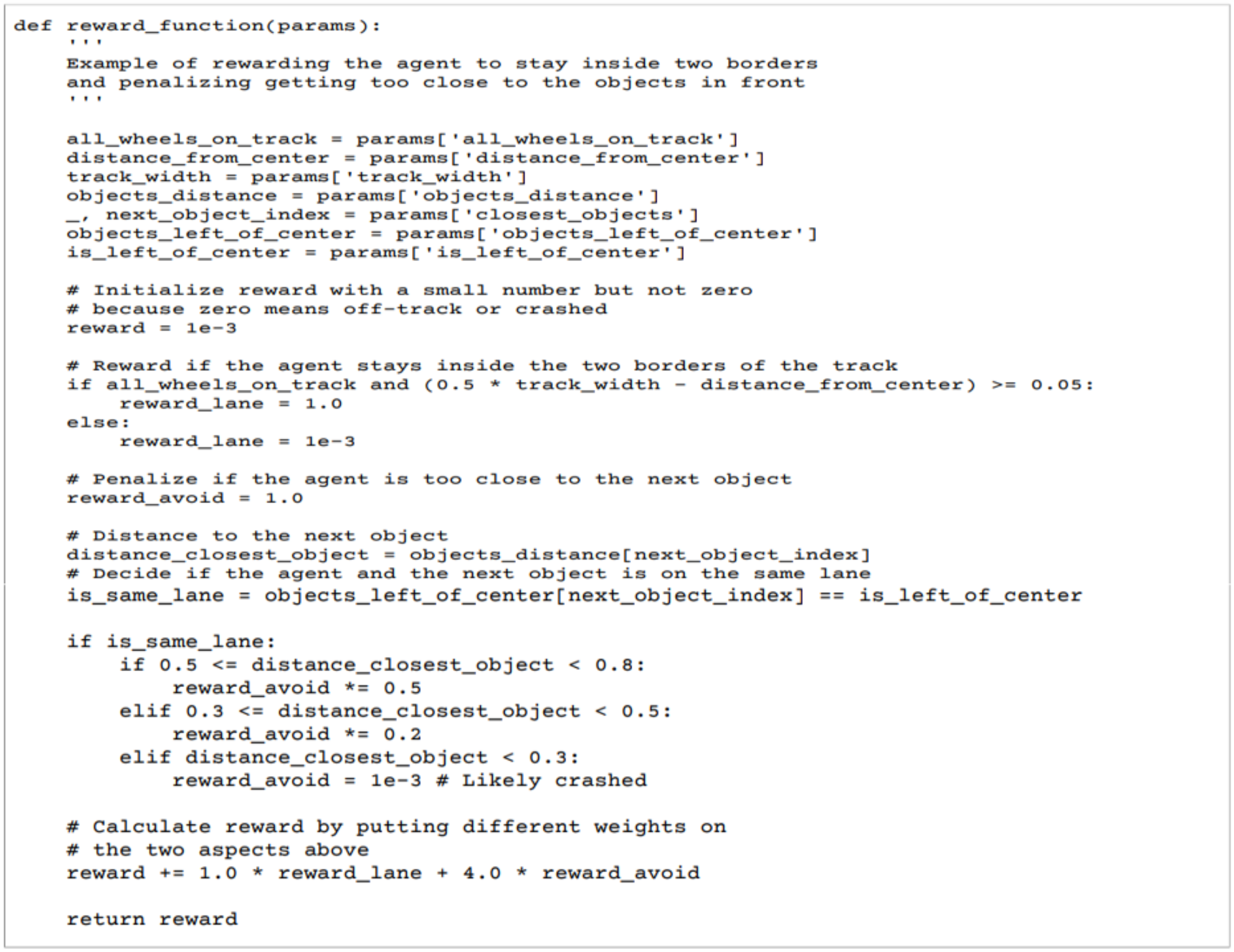
3. **Example for agent to stay inside two borders in time trials**: This simple example offers high rewards to the agent if it stays inside two borders of the track, letting the vehicle itself to find out the best track to finish the lap. Though this program is easy but it will take longer to converge.

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

    # Read input parameters
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']

    # Give a very low reward by default
    reward = 1e-3

    # Give a high reward if no wheels go off the track and
    # the car 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
```

4. **Example to Abstain from Zig-Zagging in Time Trials:** This example depicts the program to incentivize the agent if it follows center line but punishes when it steers a lot in the track while moving ahead. It will help the agent preventing from zig-zag behavior. The agent will learn to drive in a smooth way in simulator keeping the same behavior when deployed in real world scenario.

```python
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 = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    # Steering penality 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)
```

5. Example to Stay On One Lane without Crashing into Obstacles or Moving Vehicles: The reward function used in this example rewards vehicle to stay amid track borders. It gives penalties to the agent if it gets too close to object placed in front of it. Agent is allowed to move from one lane to another avoiding crashes. Total reward is calculated by adding rewards and penalties at the end. Example given below provides more weight to "penalty term" for agent to focus more on protection by evading crashes.



```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside two borders
    and penalizing getting too close to the objects in front
    '''

    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    objects_distance = params['objects_distance']
    _, next_object_index = params['closest_objects']
    objects_left_of_center = params['objects_left_of_center']
    is_left_of_center = params['is_left_of_center']

    # Initialize reward with a small number but not zero
    # because zero means off-track or crashed
    reward = 1e-3

    # Reward if the agent stays inside the two borders of the track
    if all_wheels_on_track and (0.5 * track_width - distance_from_center) >= 0.05:
        reward_lane = 1.0
    else:
        reward_lane = 1e-3

    # Penalize if the agent is too close to the next object
    reward_avoid = 1.0

    # Distance to the next object
    distance_closest_object = objects_distance[next_object_index]
    # Decide if the agent and the next object is on the same lane
    is_same_lane = objects_left_of_center[next_object_index] == is_left_of_center

    if is_same_lane:
        if 0.5 <= distance_closest_object < 0.8:
            reward_avoid *= 0.5
        elif 0.3 <= distance_closest_object < 0.5:
            reward_avoid *= 0.2
        elif distance_closest_object < 0.3:
            reward_avoid = 1e-3 # Likely crashed

    # Calculate reward by putting different weights on
    # the two aspects above
    reward += 1.0 * reward_lane + 4.0 * reward_avoid

    return reward
```

4.2 Exploring Action Space to Train Robust Model

It is a general rule that, training of a model must be robust enough to deploy it in as much environments as possible. Robust model can be applied to wide range of tracking shapes and environmental conditions. It is not "smart" as its reward functions do not have ability to be comprised of "explicit environment-specific knowledge", else model will probably applicable to only trained tracks. "Explicitly incorporating environment-specific information into the reward function amounts to feature engineering".

With feature engineering, training time can be reduced and can be helpful in providing solutions tailor created for a particular environment. For general model training, user should abstain from attempting more feature engineering.

To make the model robust enough to be deployed in any environment, one way for it is to explore the "action space" covering the actions taken by the agent. Another way is to use hyper-parameters for training algorithms.

While training AWS DeepRacer model, action (a) is the combination of speed (t m/s) and steering angle (s in degrees). Agent action space defines its “ranges of speed” and “steering angle” that can possibly be taken by the agent. For discrete action space of “m” number of speeds, (v1……vn) and “n” number of steering angles, (s1, .., sm), “m*n” possible actions are there in action space.

a_1: (v_1, s_1)

...

a_n: (v_1, s_n)

...

$a_{(i-1)*n+j}$: (v_i, s_j)

...

$a_{(m-1)*n+1}$: (v_m, s_1)

...

a_{m*n}: (v_m, s_n)

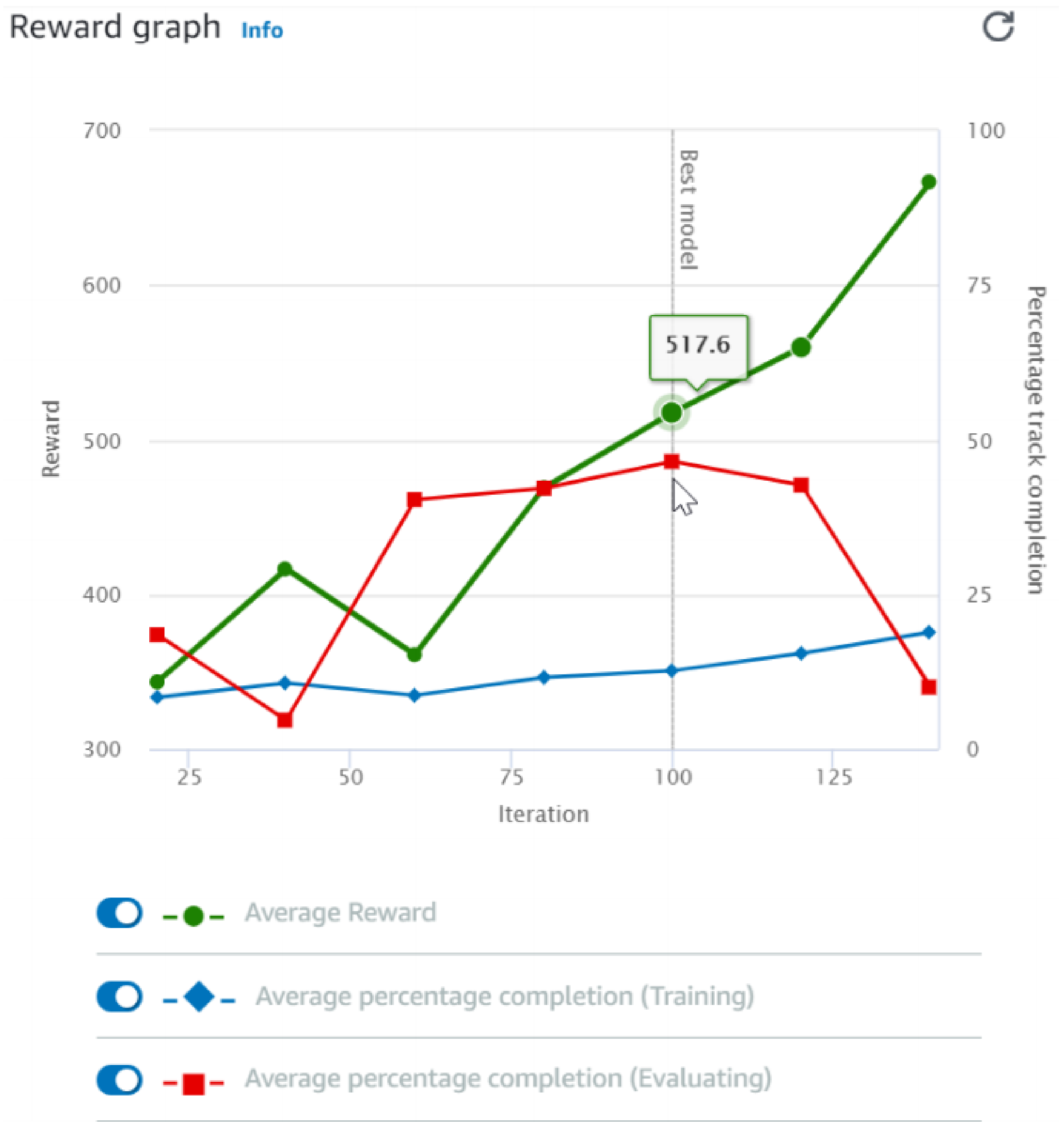
Actual values of (vi, sj) depend on ranges of vmax and |smax| and are not uniformly distributed.

On beginning of each training iteration of AWS model, user must specify the above mentioned variables first. Based on these, AWS services builds the actions that vehicle can select while training. These created actions are not “uniformly distributed” on the action space. Large number of actions and ranges provide the vehicle with more room to react to varying track conditions, for instance, curved track with irregular turning angles or directions. With large number of options availability to agent, more readily agent can handle the curvy or changing track making the vehicle (agent) more robust to every environment.

When the action space is too large, performance of training can suffer, as it will then take longer to explore action space. User must be sure to balance advantages of general applicability of model against training performance requirements. This optimization includes the systematic experimentation.

4.3 Examining DeepRacer Training Job Progress

On initializing the training job, user can determine “training metrics” of rewards and track “completion per episode” to establish the performance of training job of the model. On AWS DeepRacer console, metrices are displayed in “Reward Graph” as illustrated below.



4.4 Cloning Trained Model to Start New Training Pass

Cloning of a trained model using AWS DeepRacer Console is possible by following the steps mentioned below:

1. At first, sign in to AWS DeepRacer console.
2. On "Models page", select trained model and then select "Clone" from drop down menu list of "Action".
3. For "Model details", follow these:
    a. Type "**RL_model_1**" in "Model name".
    b. Give the brief description of the model to be cloned in "Model description – optional".

4. Select option for another track option for "Environment simulation".
5. Choose any of the available reward function examples for model "Reward function", and "Modify" it.
6. Expand the "Algorithm settings" and choose varying options. For instance, change Gradient descent batch size value from 33 to 64 or user can increase the "Learning rate" to speed up the training.
7. Experiment with different choices of "Stop conditions".
8. Click on the "Start training" button to begin new round of training.

To train a robust ML model, user must conduct a systematic experimentation for receiving best solutions.

4.5 Evaluating AWS DeepRacer Model in Simulations

Evaluating the model is actually analyzing the performance of the trained model in simulation. The steps taken while evaluating model are as follows:

1. Configure and start an evaluation job.
2. See the "in progress" evaluations while job is running. This is done with the help of using AWS DeepRacer simulator.
3. Examine the evaluation summary when evaluation job is completed. Termination of evaluation job can be done any time while it is in progress.
4. Submit this evaluation results to the AWS DeepRacer leaderboard. Leaderboard ranking will let the user know about the performance of the model against other models in the race.

4.6 Logging AWS DeepRacer Events to CloudWatch Logs

For analytical purposes, the AWS DeepRacer reports some runtime events to "CloudWatch Logs" while training and evaluation. These events are logged in under "job-specific log streams", such that,

- **For training job:** under "*/aws/sagemaker/TrainingJobs*" log group.
- **For simulation job:** under "*/aws/robomaker/SimulationJobs"* log group.
- **For evaluation job:** under "*/aws/deepracer/ leaderboard/SimulationJobs*" log group.
- **For reward function execution:** under "*/aws/lambda/AWS-DeepRacer-Test-Reward-Function"* log group.

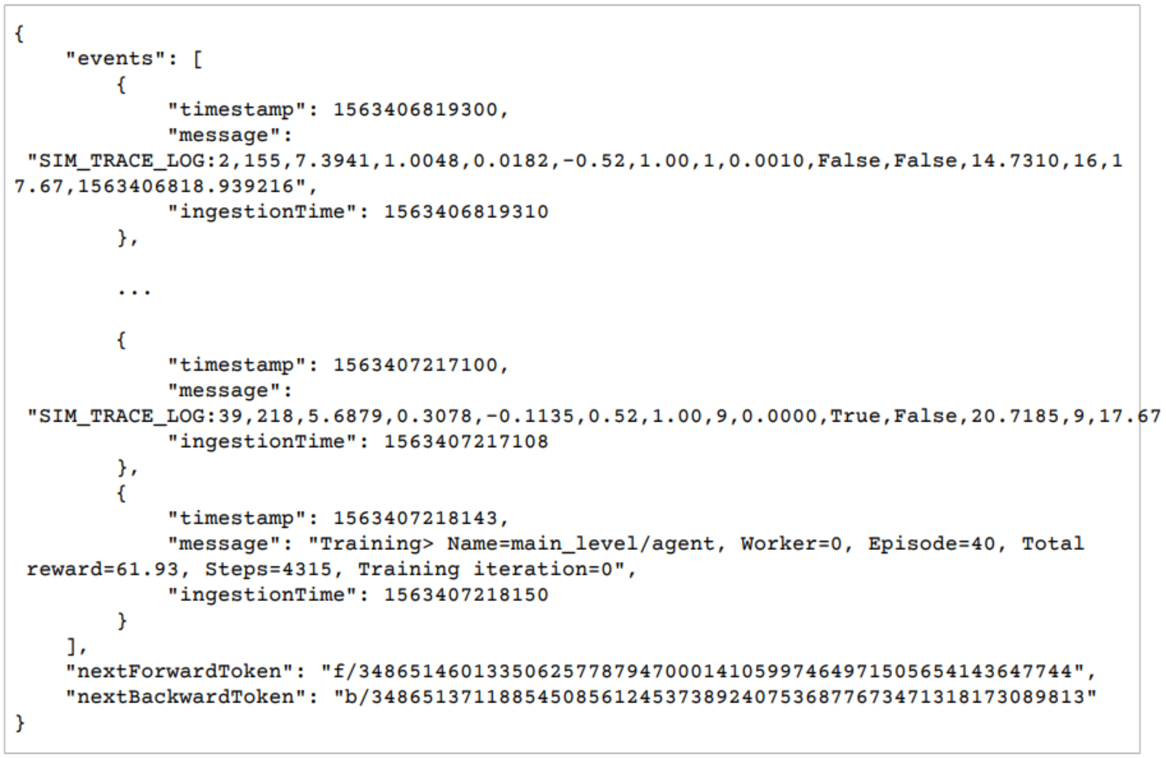
To get the access of AWS DeepRacer logs, user can make use of the "CloudWatch console", "AWS CLI" and "AWS SDK". Codes to access these logs for each category are mentioned below.

4.6.1 To assess AWS DeepRacer logs using the AWS CLI

1. Open the access window and type the following.

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

2. The output came out to be similar to this one.



```
{
    "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
            "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"
}
```

4.6.2 To assess AWS DeepRacer logs in the CloudWatch Logs console

1. Sign in to “CloudWatch console” and click on “Logs” from the navigation pane.
2. Choose a suitable “log group”.
3. Choose “log stream” to open the log file.

4.7 Optimization of training AWS DeepRacer Model for Real Environment

There are many factors that affect the performance of model in real-world such as: action space, hyper-parameters, reward functions, vehicle calibration and real world track. It must be kept in mind that simulations are only an approximation for creating real world scenario. To train and improve the performance of model for solid real-world requires several iterations of exploring reward functions, action space, hyper-parameters, evaluations and simulation testing in real environment.

Testing of simulation for real world environment includes “simulation-to-real world (sim2real) function” that can be cumbersome. To avoid the challenges of (sim2real) function, user must take care about the following steps for optimization.

1. It must be sure that agent (vehicle) is well calibrated.
2. Always test the vehicle first with the default model.
3. It must be made sure that model is fully functional and working in the simulations.
4. Try not to over train the model, otherwise it will cause overfitting to the model.
5. Always consider multiple models from different iteration.
6. While testing the model, at first keep the agent speed low and then gradually increase the speed.

7. While testing for real world scenario, test and validate model performance by making it to start at any point on the track.
8. Always start testing the model by placing it on a straight path first, rather than the curved path/ track.
9. Observe the behavior of the vehicle about when it takes only one kind of actions.
10. Observe the inconsistent behavior of the agent (vehicle).
11. Observe the ability of vehicle to correct its path on a track.
12. Observe the vehicle's type of turn, sudden turn and off track turns.

5 Train and Evaluate the AWS DeepRacer Model using Amazon SageMaker Notebooks

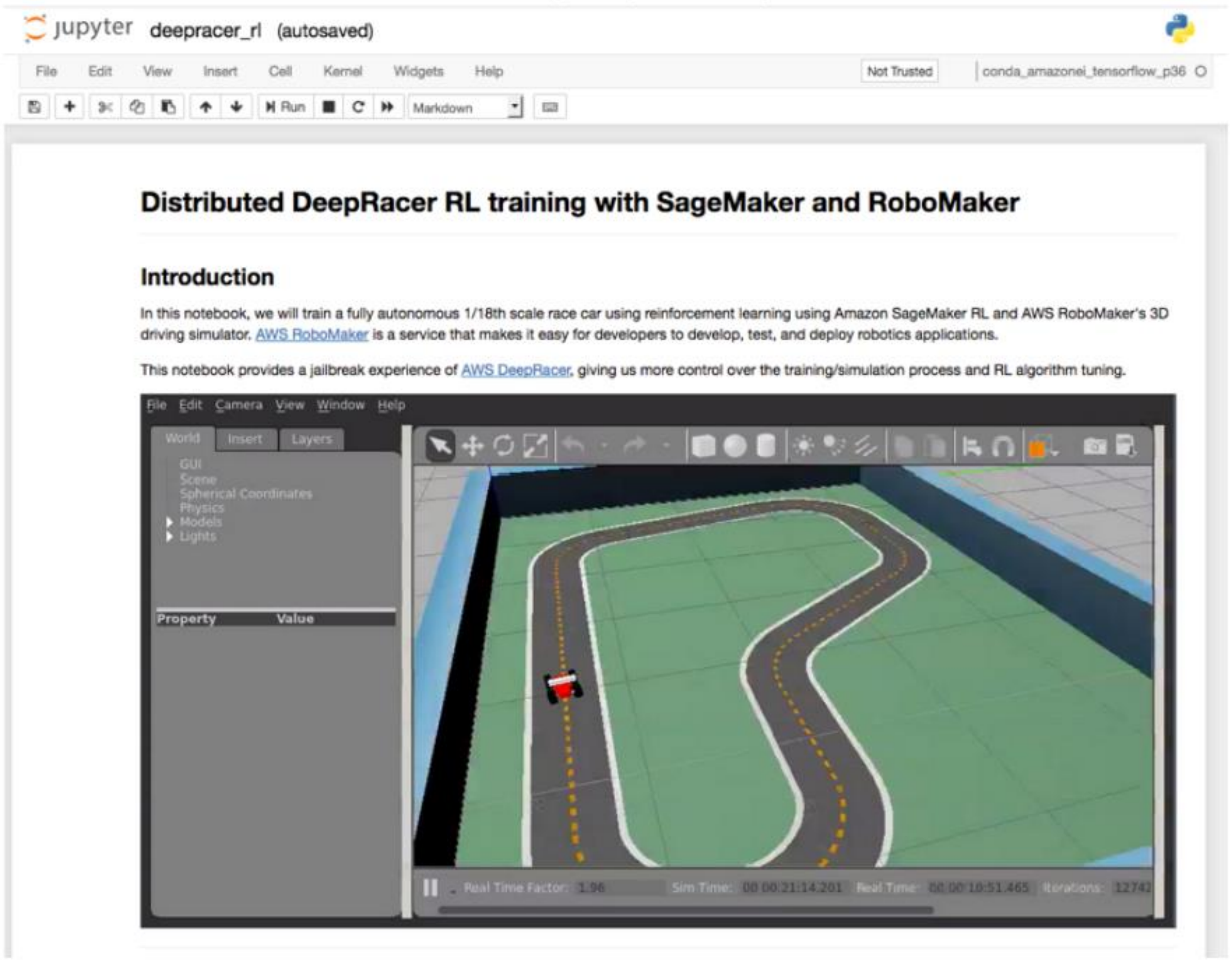
AWS DeepRacer console offers the integrated experience for training and evaluating the AWS models as it uses "Amazon SageMaker & AWS RoboMaker" including detailed "reinforcement learning (RL)" tasks. This section guides with the step by step illustration of these tasks.

5.1 Creating Amazon SageMaker Notebook

To directly train the AWS model in "Amazon SageMaker" following steps are taken into consideration.

1. Sign in the "Amazon SageMaker Console" at (https://console.aws.amazon.com/sagemaker) and select the supported region.
2. First select "Notebook instances" and then click on the "Create Notebook Instance" from navigation pane.
3. On "create Notebook instance" page, do these steps:
    a. Type name e.g. (my-deepracer-model)
    b. Click "Create new role" from the drop down menu list of "**IAM, Enter the custom IAM role ARN, or Use existing role"** and then follow instructions.
    c. Leave the default choices for all other options and then choose "Create notebook instance".
4. Wait for "notebook instance's Status" to change from "Pending" to "InService", then click on the "Open JUPYTER".
5. On "Jupyter" page (home page of recently created new notebook), make these steps:
    a. Choose the "SageMaker Examples" tab.
    b. Expand the "Reinforcement Learning" example group from example collection.
    c. For this particular exercise, click on "Use" next to the "deepracer_rl.ipynb" item.
    d. On "Create a copy in your home directory" dialog, select "Create copy".

Here, the notebook instance is executing and user can start to train the model. User is charged for running "notebook instances" at this point, so it is better to shut down the instance to avoid charging.


jupyter deepracer_rl (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
conda_amazonei_tensorflow_p36
Run
Markdown

Distributed DeepRacer RL training with SageMaker and RoboMaker

Introduction

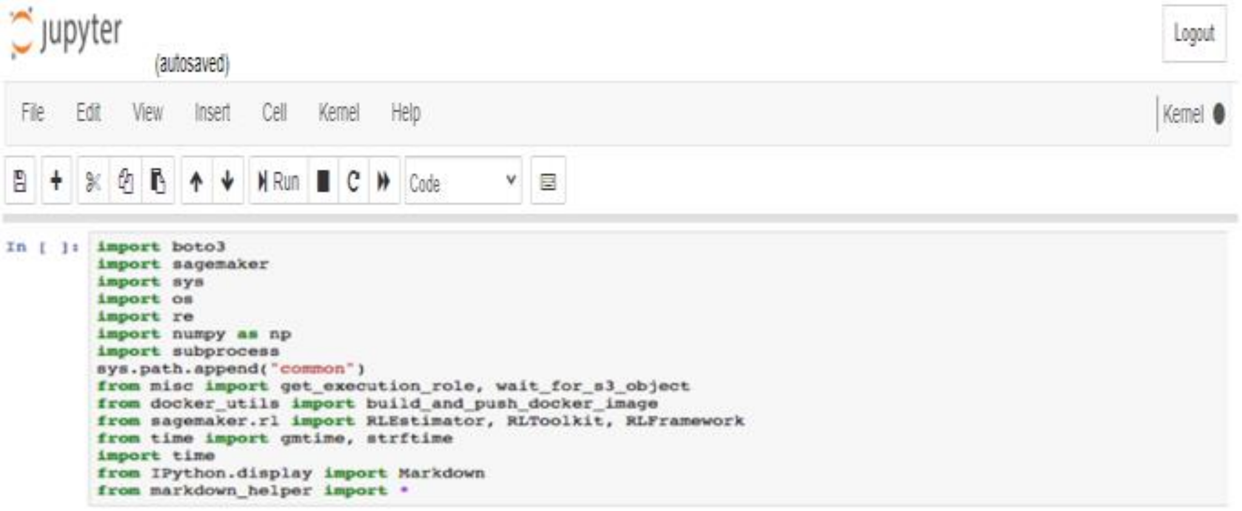
In this notebook, we will train a fully autonomous 1/18th scale race car using reinforcement learning using Amazon SageMaker RL and AWS RoboMaker's 3D driving simulator. AWS RoboMaker is a service that makes it easy for developers to develop, test, and deploy robotics applications.

This notebook provides a jailbreak experience of AWS DeepRacer, giving us more control over the training/simulation process and RL algorithm tuning.

5.2 Initializing Amazon SageMaker Notebook Instance

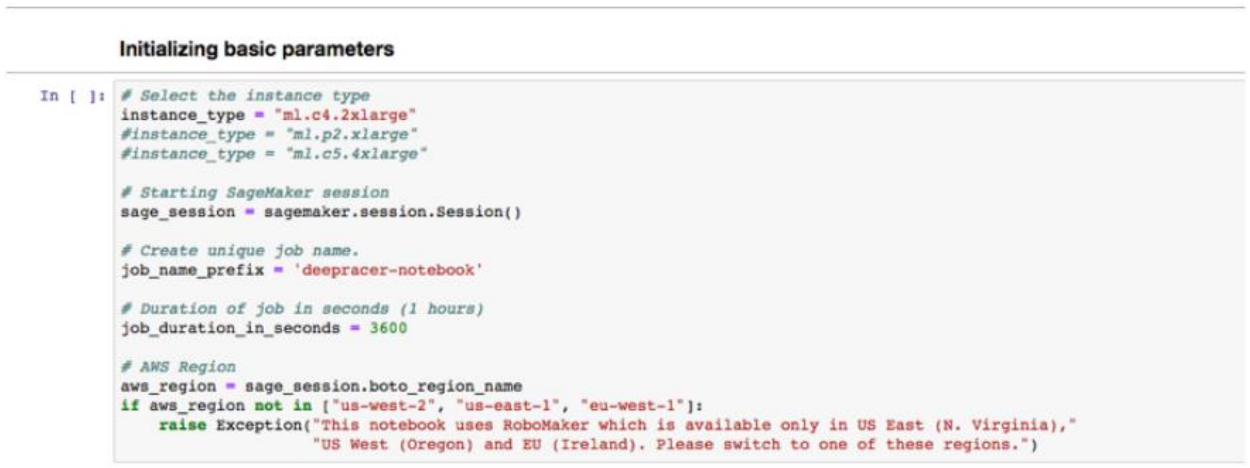
To initialize the Amazon SageMaker notebook, follow the instructions mentioned below:

1. At first, to start training the model using "Jupyter", import the libraries as shown in Figure below and click on "run" button in the bar above the notebook.



```
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 *
```

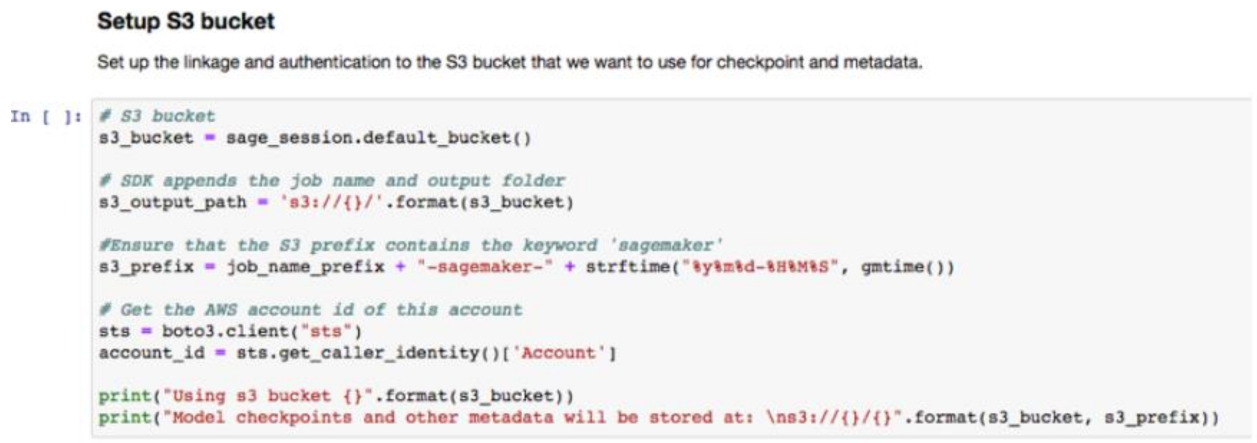
2. Initialize the basic parameters by running “Initializing basic parameters” code block as depicted in figure.



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 = 'deepracer-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.")
```

3. For setting up the training output storage, select code block below “Setup S3 bucket’, and click “Run” from notebook instance menu bar.

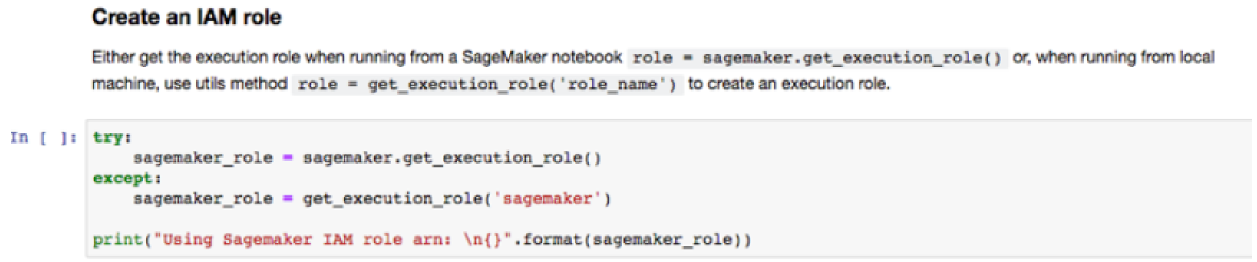


Setup S3 bucket

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))
```

4. For setting up the proper permissions for notebook instance to access “S3 storage” for output by “Amazon SageMaker”, run code cell under “Create an IAM role”.



Create an IAM role

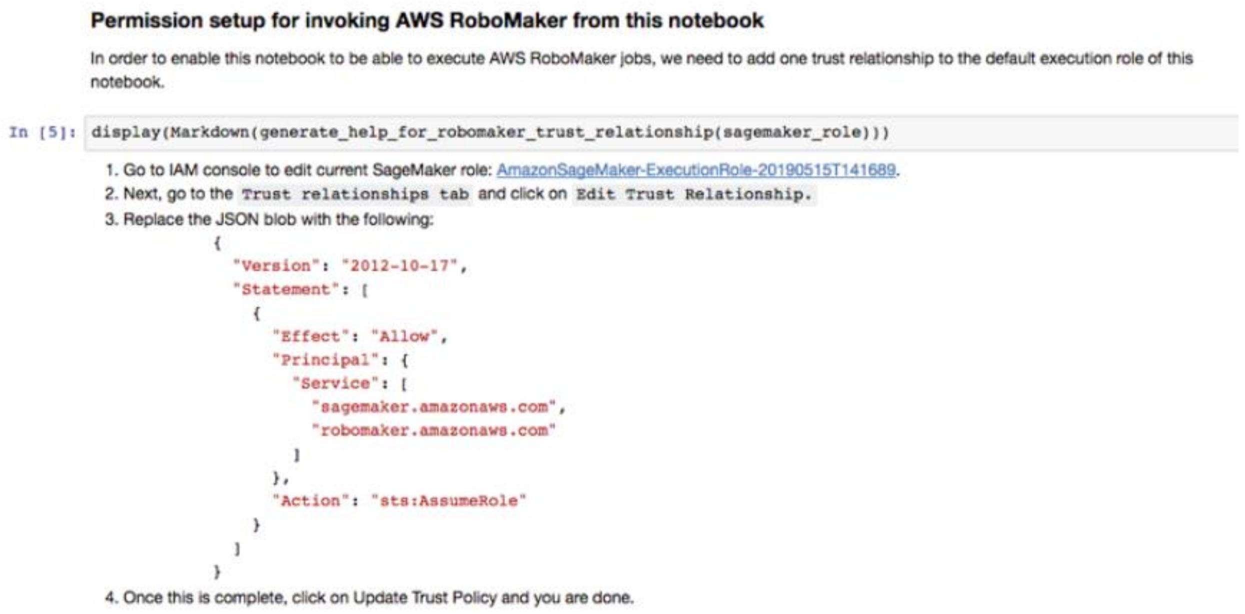
Either get the execution role when running from a SageMaker notebook `role = sagemaker.get_execution_role()` or, when running from local machine, use utils method `role = get_execution_role('role_name')` to create an execution role.

```
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 the simulation is completed, the code block generates the new “IAM role” comprising the following “IAM policy”.

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

5. To invoke the "AWS RoboMaker" to execute training environment, run code cell under "Permission setup for invoking AWS RoboMaker" and follow instructions to add "robomaker.amazonaws.com" as another "trusted entity" of the IAM role generated before.



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.

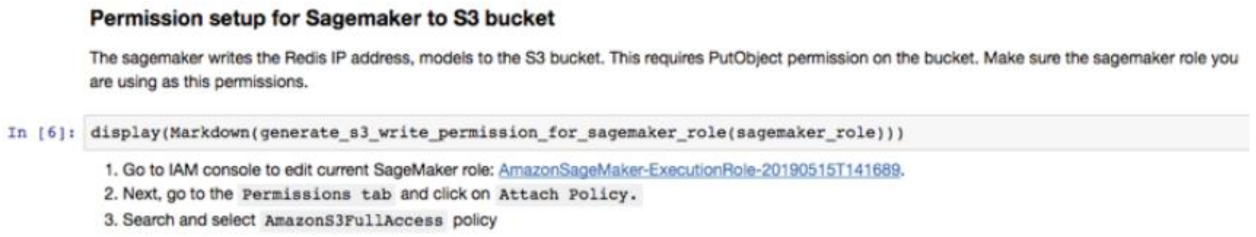
```
In [5]: display(Markdown(generate_help_for_robomaker_trust_relationship(sagemaker_role)))
```

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 assess "S3 storage", run code cell under "Permission setup for SageMaker to S3 bucket" and follow instructions as guided to attach "AmazonS3FullAccess policy" to "IAM role" created beforehand.



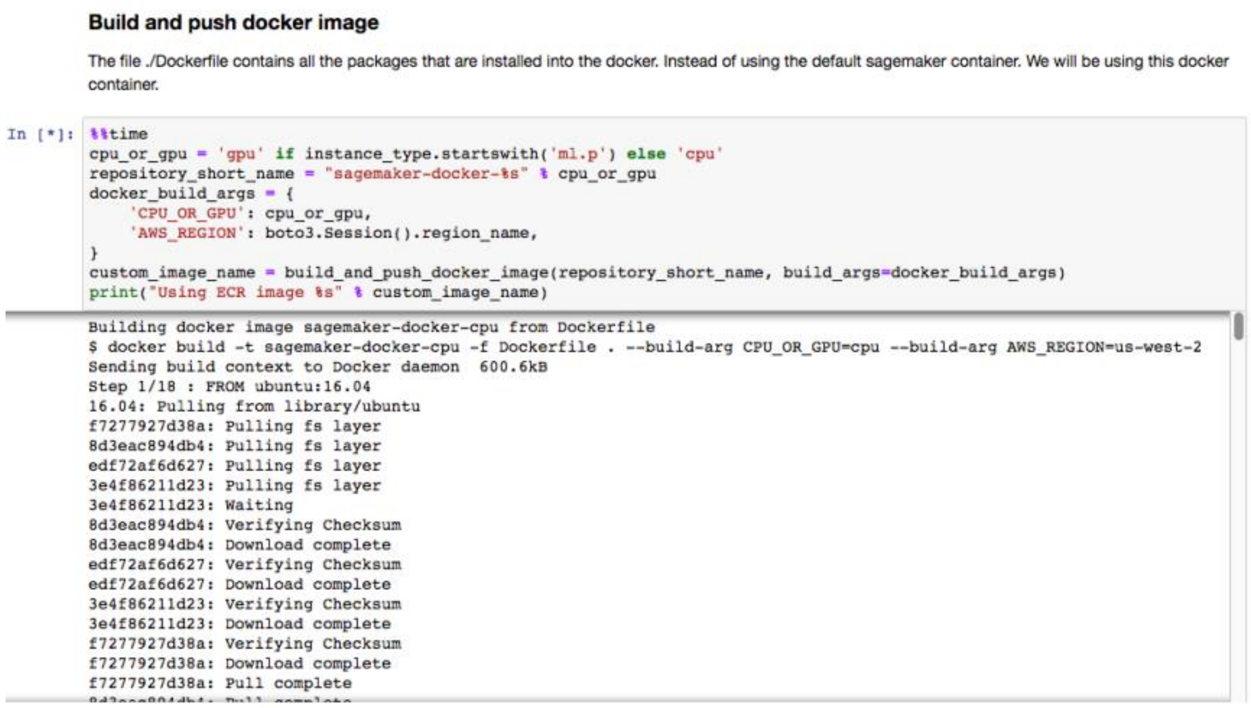
Permission setup for Sagemaker to S3 bucket

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 as 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. Run "build and push docker image" to provide a docker container for executing the training and evaluation job of the model. It will take longer to finish.

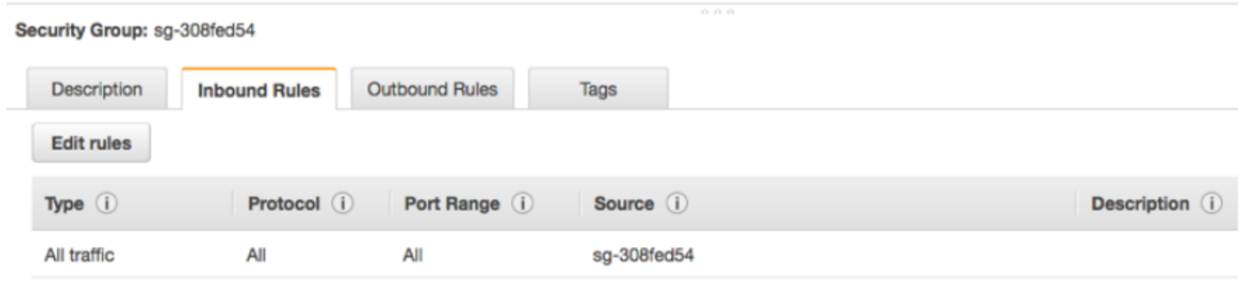

Build and push docker image

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
```

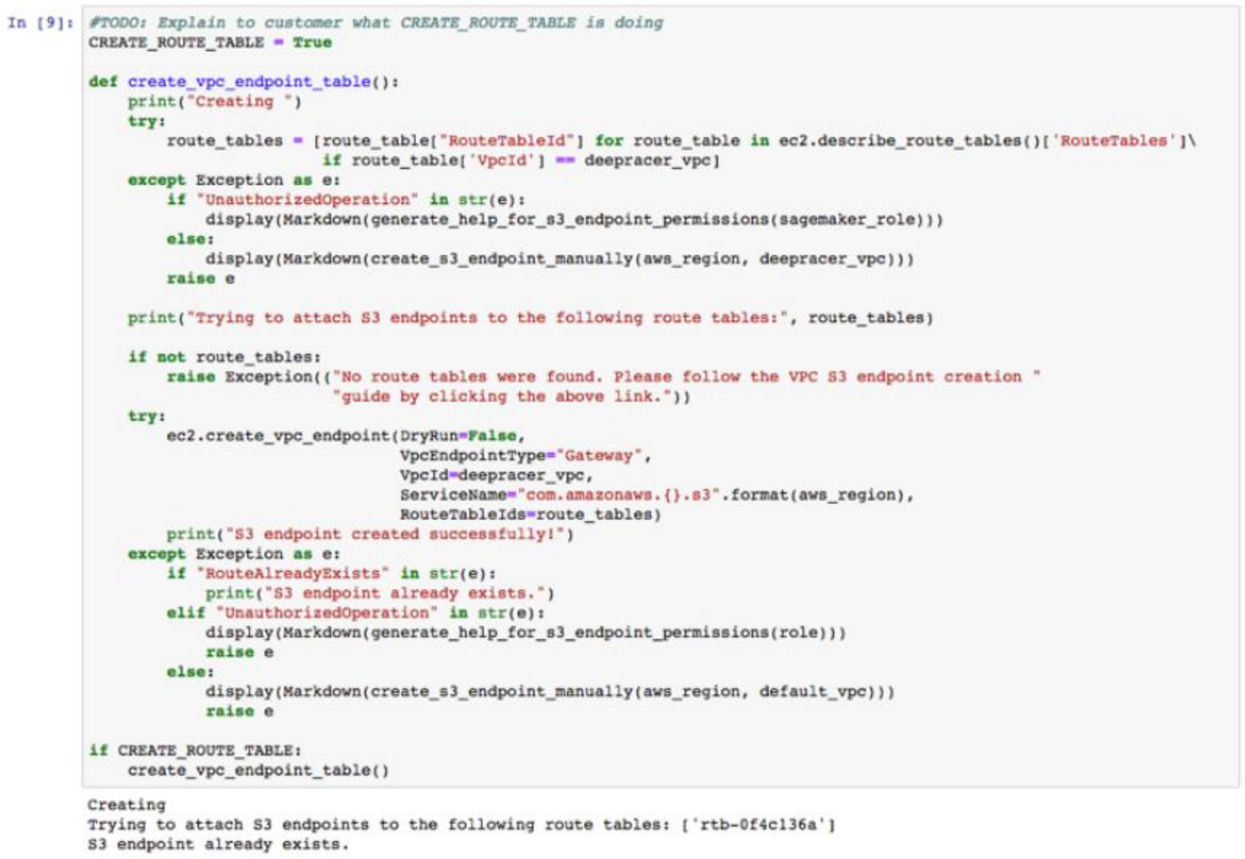
8. For enabling "VPC mode" for "Amazon SageMaker and AWS RoboMaker" to interconnect with one another over the network, run "Configure VPC". Jupyter notebook instance uses "default VPC, security group, and subnets" for configuring VPC mode by default.


Security Group: sg-308fed54

Type	Protocol	Port Range	Source	Description
All traffic	All	All	sg-308fed54	

9. For enabling "SageMaker training job" to access "S3 resources", run code of "Create Route Table" to build "VPC S3 endpoint".

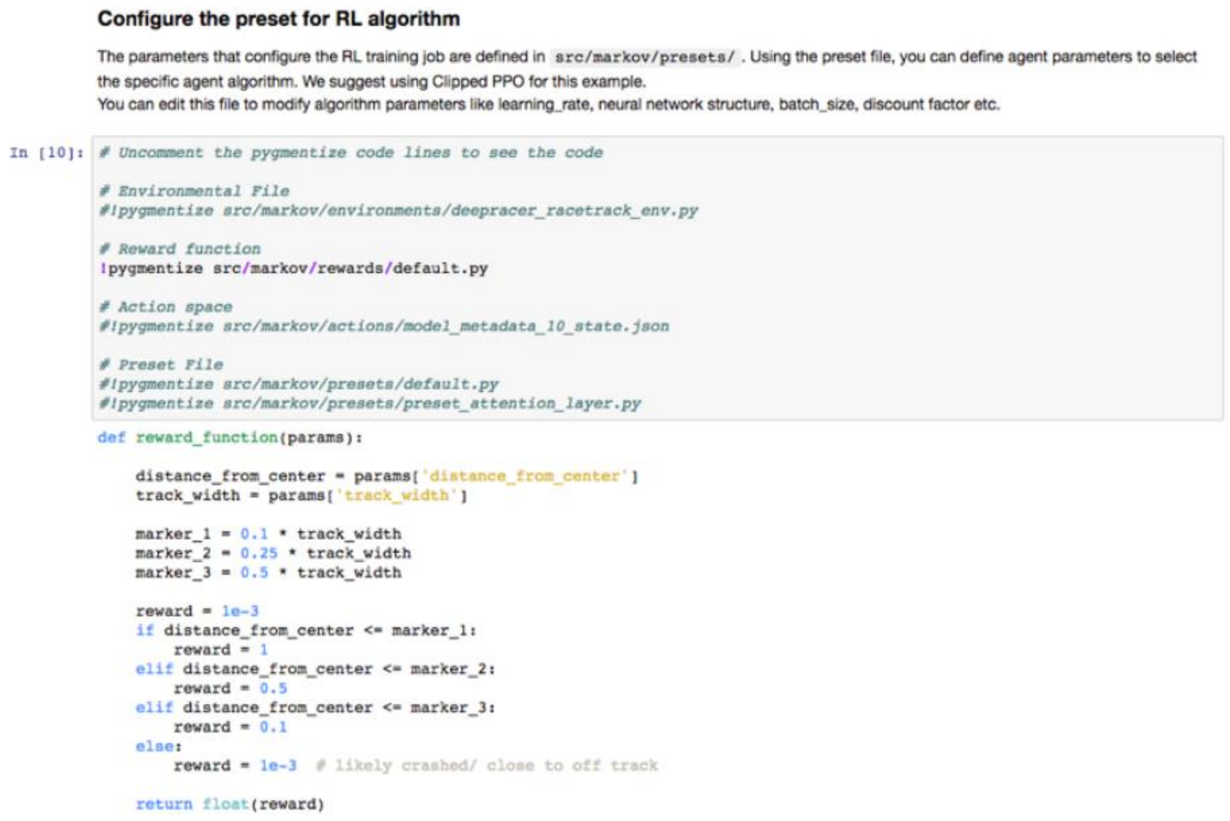
After this, initialization of the training model is completed and user will be ready for setting up the environment.



```
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.
```

5.3 Setting Up the Training Environment

For setting up the training environment to train the “AWS DeepRacer” includes selection of race track, reward function, action space and hyper-parameters. As a new user, notebook make use of default setting. To see the default settings, run the code by uncommenting the relevant part of the code under “Configure the preset for RL algorithm”. Run code as illustrated below:


Configure the preset for RL algorithm

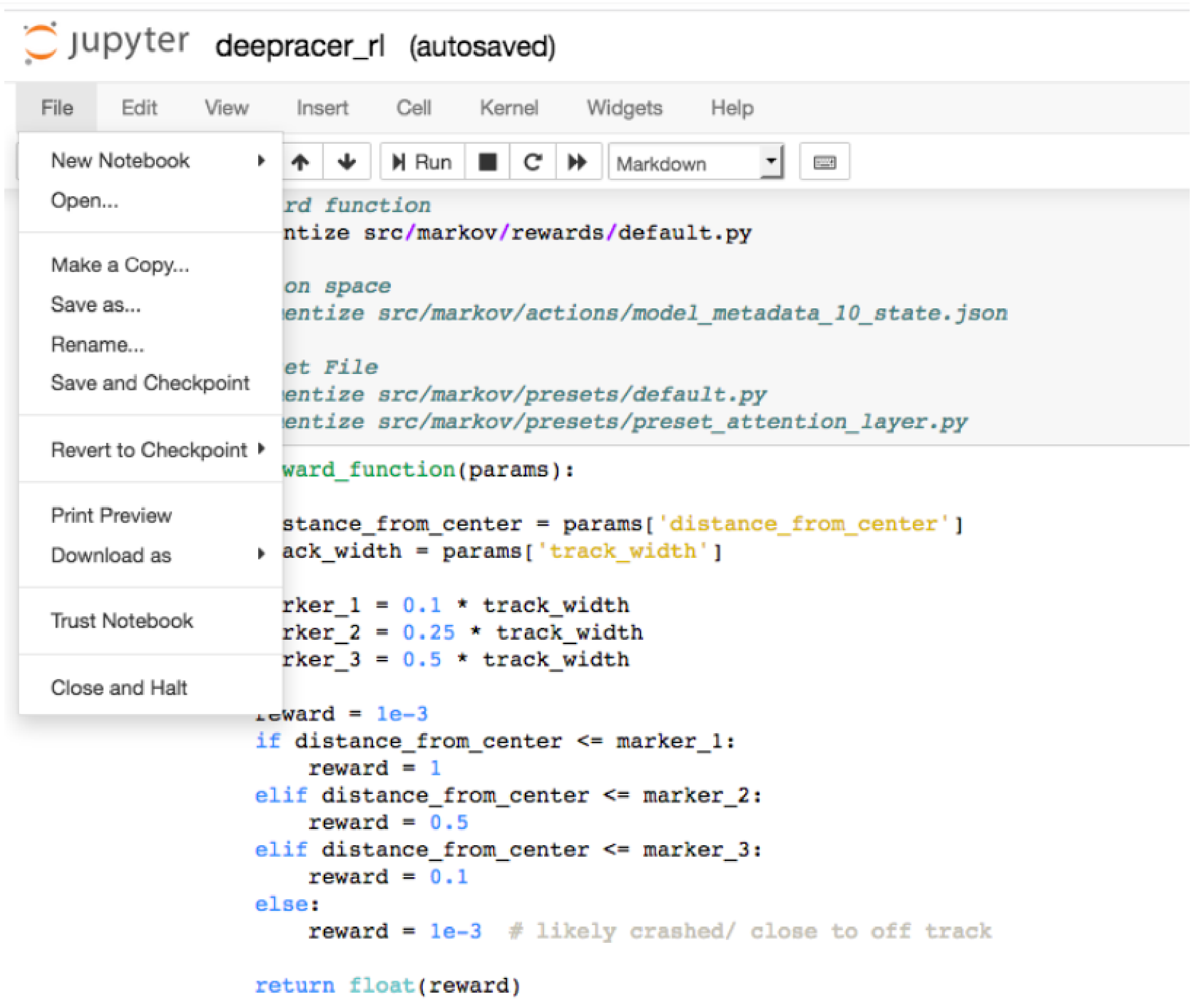
The parameters that configure the RL training job are defined in `src/markov/presets/`. Using the preset file, you can define agent parameters to select the specific agent algorithm. We suggest using Clipped PPO for this example.
You can edit this file to modify algorithm parameters like learning_rate,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

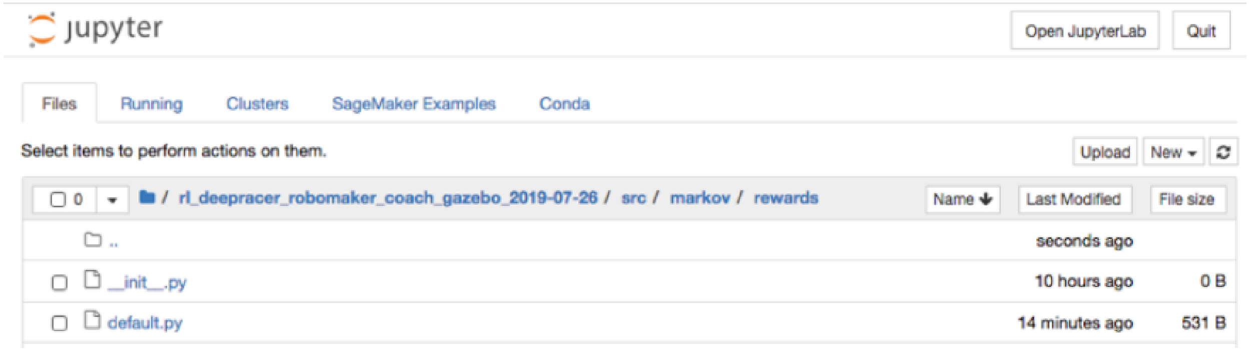
If user keep on working with the default setting, then just copy the same file, rename this file and its directory and modify it by following the procedure below:

1. Click on the “file” menu and select “Open”.



```
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 “src/markov/rewards” folder and select “default.py” to open file.

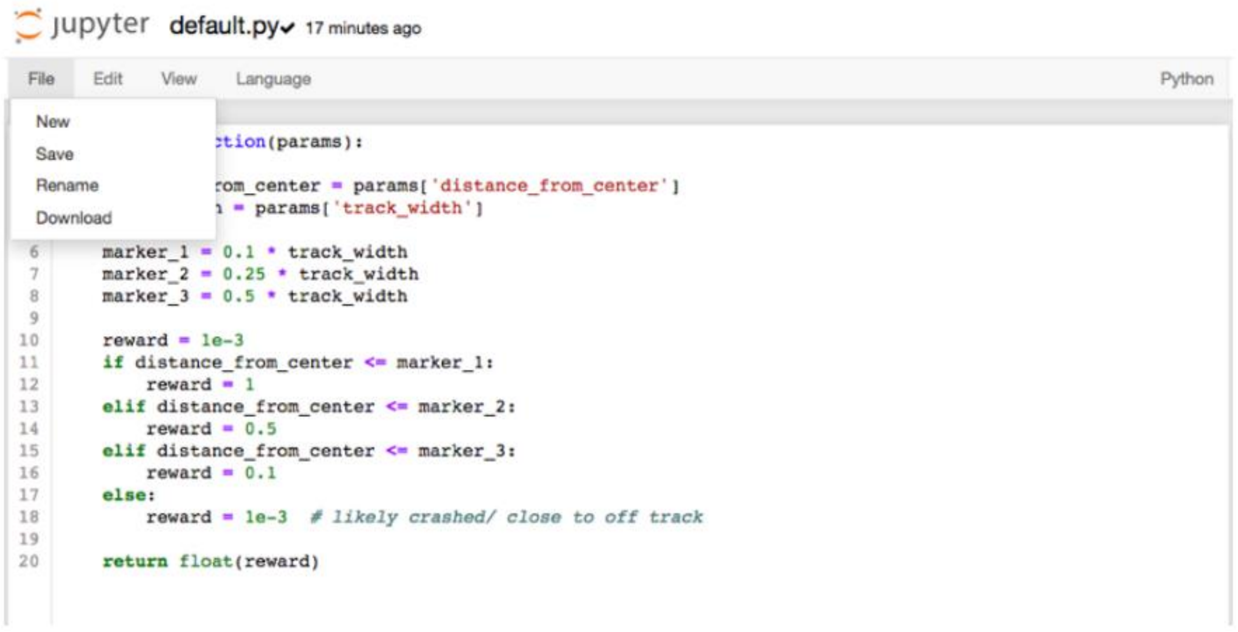


Select items to perform actions on them.

/ rl_deepracer_robomaker_coach_gazebo_2019-07-26 / src / markov / rewards

..	seconds ago	
__init__.py	10 hours ago	0 B
default.py	14 minutes ago	531 B

3. Edit the file name accordingly and save it as: “File > Save” to save the update.

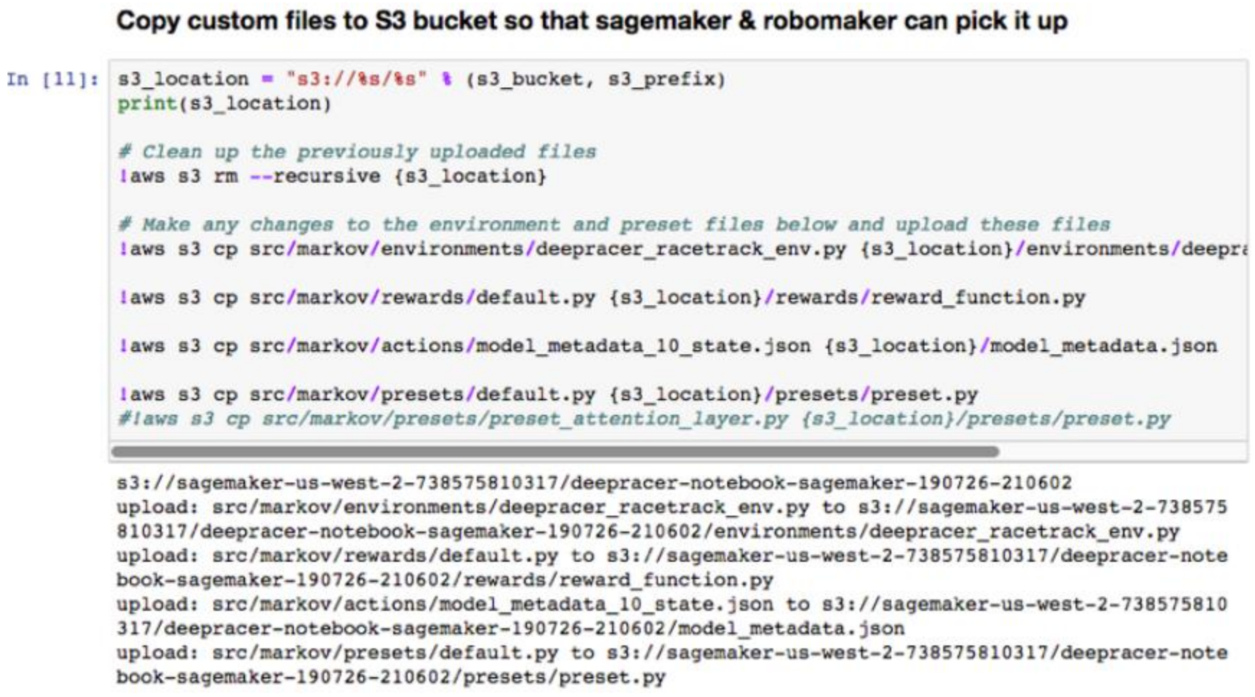


```
             :tion(params):

              :om_center = params['distance_from_center']
              ı = 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)
```

5.4 Training the AWS DeepRacer model

To train the AWS DeepRacer model, consider the following steps below:

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

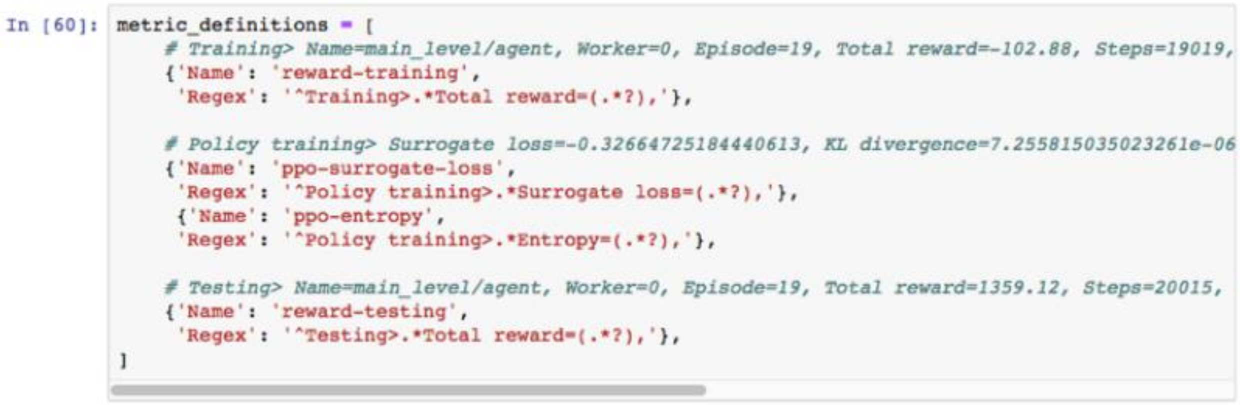


Copy custom files to S3 bucket so that sagemaker & robomaker can pick it up

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/deepra

!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
```

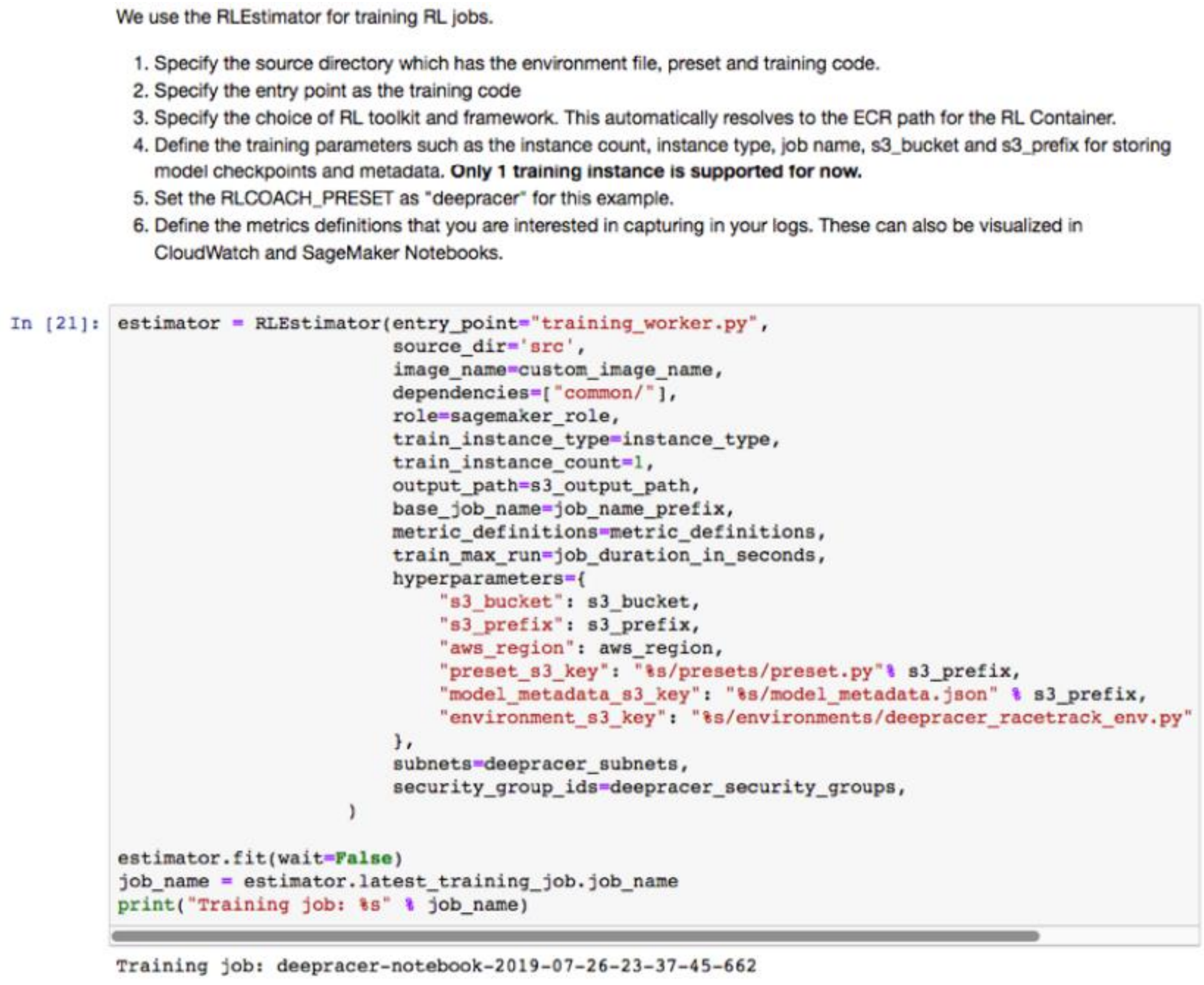
```
s3://sagemaker-us-west-2-738575810317/deepracer-notebook-sagemaker-190726-210602
upload: src/markov/environments/deepracer_racetrack_env.py to s3://sagemaker-us-west-2-738575
810317/deepracer-notebook-sagemaker-190726-210602/environments/deepracer_racetrack_env.py
upload: src/markov/rewards/default.py to s3://sagemaker-us-west-2-738575810317/deepracer-note
book-sagemaker-190726-210602/rewards/reward_function.py
upload: src/markov/actions/model_metadata_10_state.json to s3://sagemaker-us-west-2-738575810
317/deepracer-notebook-sagemaker-190726-210602/model_metadata.json
upload: src/markov/presets/default.py to s3://sagemaker-us-west-2-738575810317/deepracer-note
book-sagemaker-190726-210602/presets/preset.py
```

2. To begin "Amazon SageMaker job" to train the AWS DeepRacer model, do the following:
   a. Run first code cell under "Train the RL model using the Python SDK Script mode" to describe training metrics to observe it in either CloudWatch Logs or in 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=(.*?),'},
         ]
```

b. Run second code cell under “Train the RL model using the Python SDK Script mode” to begin Amazon SageMaker training job for the model.

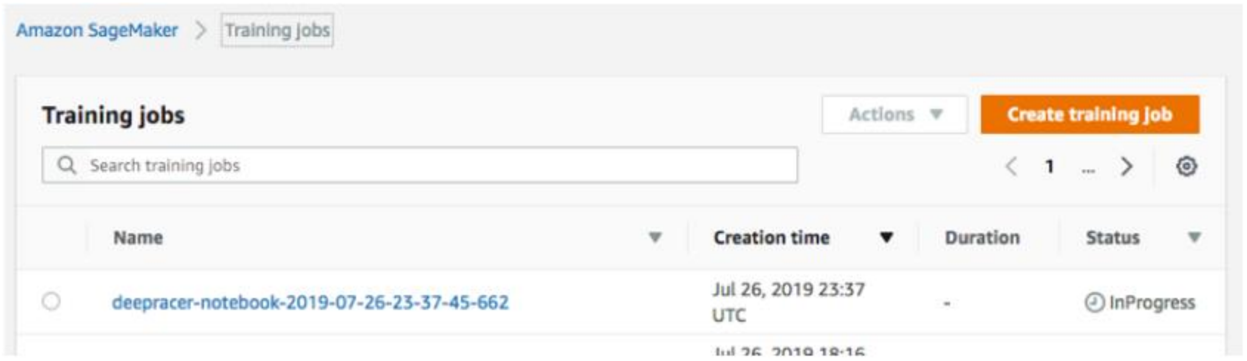


We use the RLEstimator for training RL jobs.

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
```

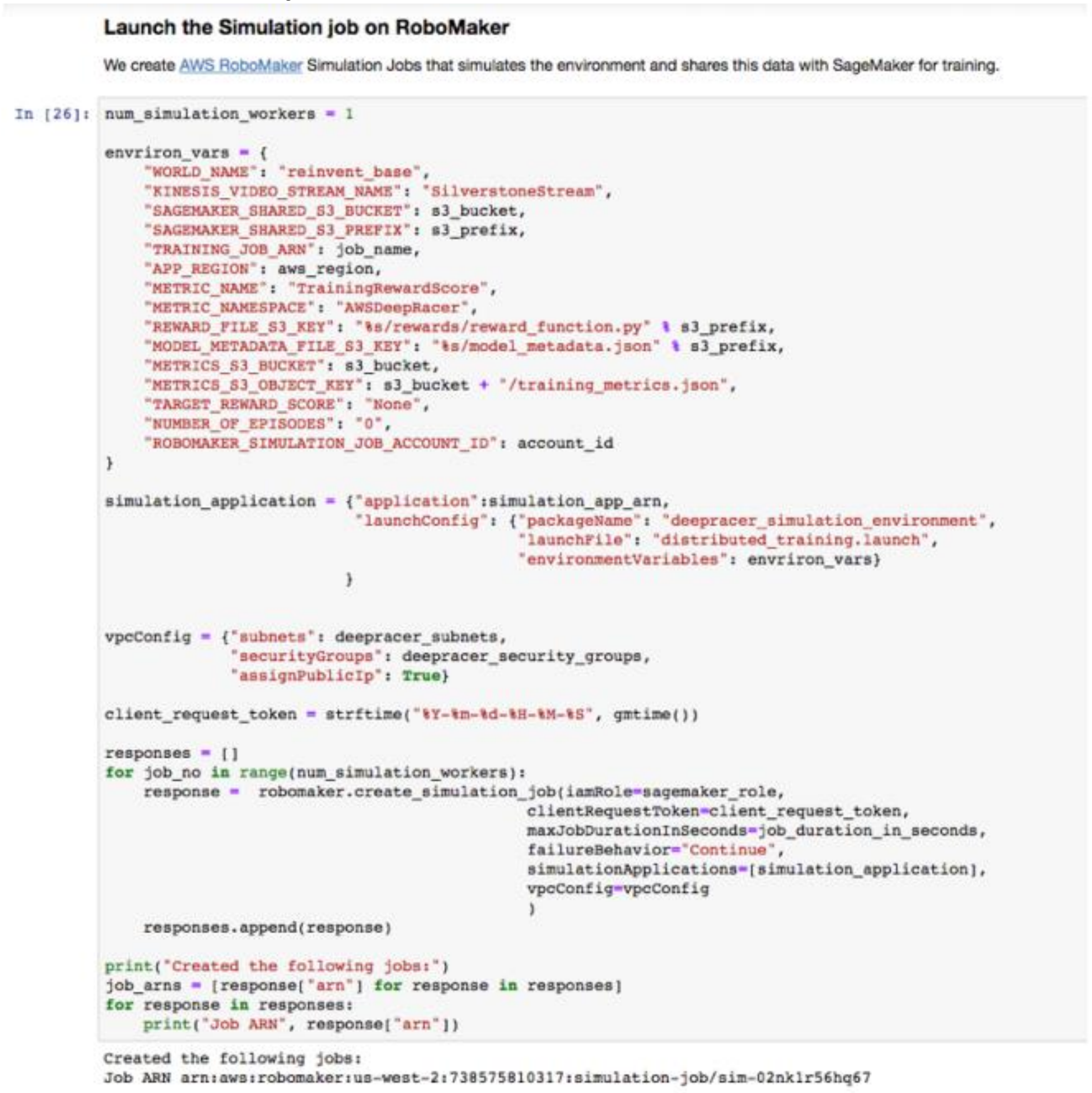
It uses “TensorFlow framework” and executes on particular EC2 compute instance type. The output enlists job name. user can track status in Amazon SageMaker.


Amazon SageMaker > Training jobs

Training jobs

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

3. To make a simulation environment in AWS RoboMaker, execute the cell under “Start the RoboMaker job and Create Simulation Application”.
4. To begin simulation on “AWS RoboMaker” and share “simulated data”, execute code under “Launch the Simulation job on RoboMaker”.

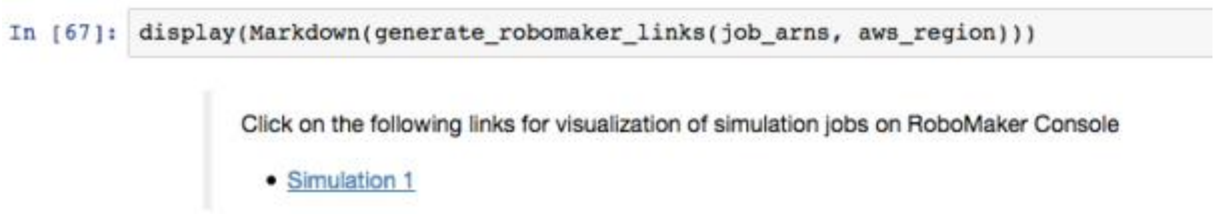

Launch the Simulation job on RoboMaker

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": "deepracer_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-02nklr56hq67
```

5. Visualize the simulations by running code cell beneath “Visualizing the simulations in RoboMaker” and select the link of “Simulation 1” from output.



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

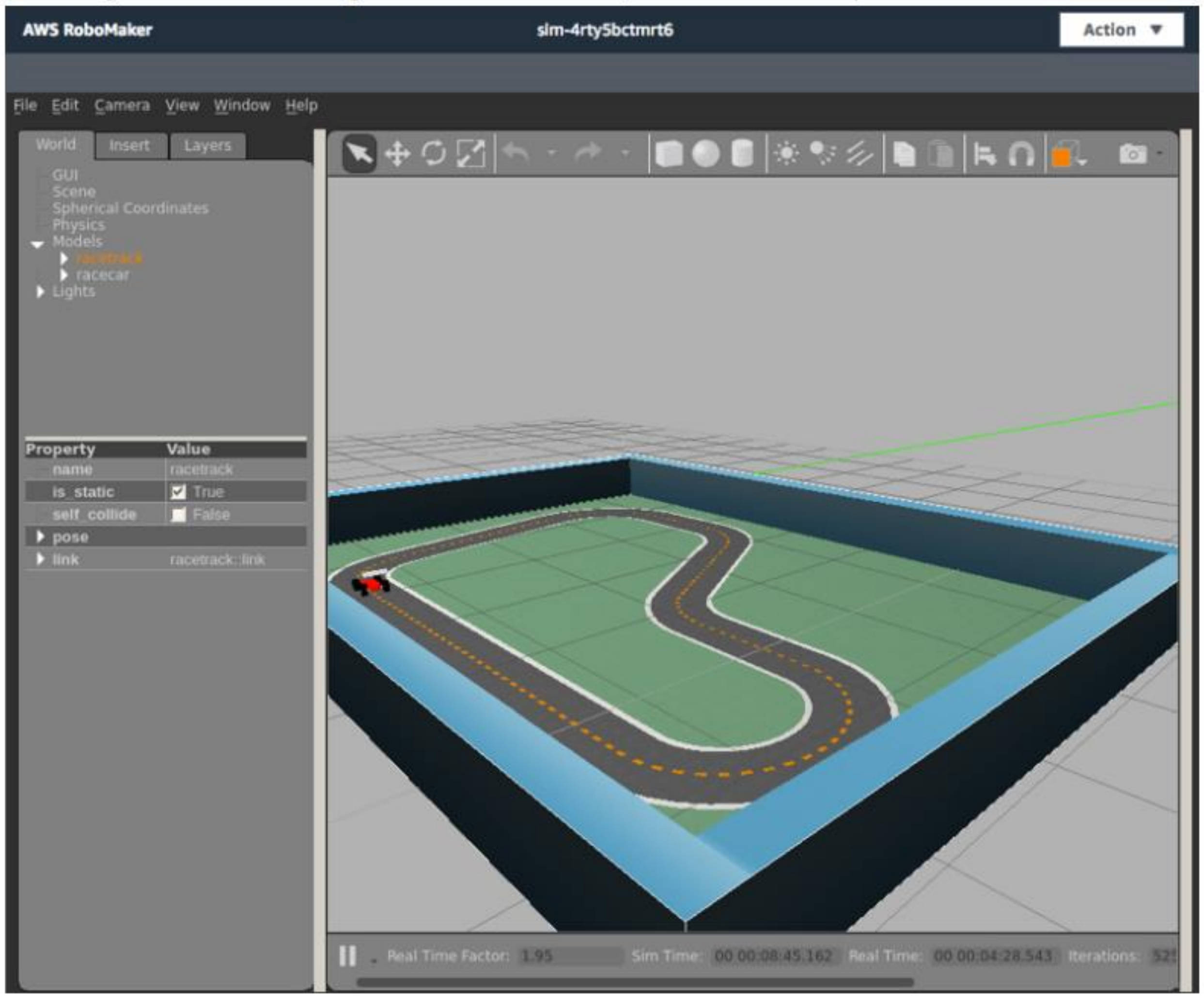
Click on the following links for visualization of simulation jobs on RoboMaker Console

- Simulation 1

When the simulation is made to execute, following visualization utilities become available on the AWS RoboMaker console:

- **Gazebo:** imitation of 3D worlds for running automatically driven vehicle in selected track.
- **rqt:** Qt-based framework and plugins for “ROS GUI” development.
- **ivis:** a “ROS visualizer” for exhibiting field of vision that is taken by vehicle's front-facing camera.
- **Terminal:** terminal application to offer access to the command line on simulation host.

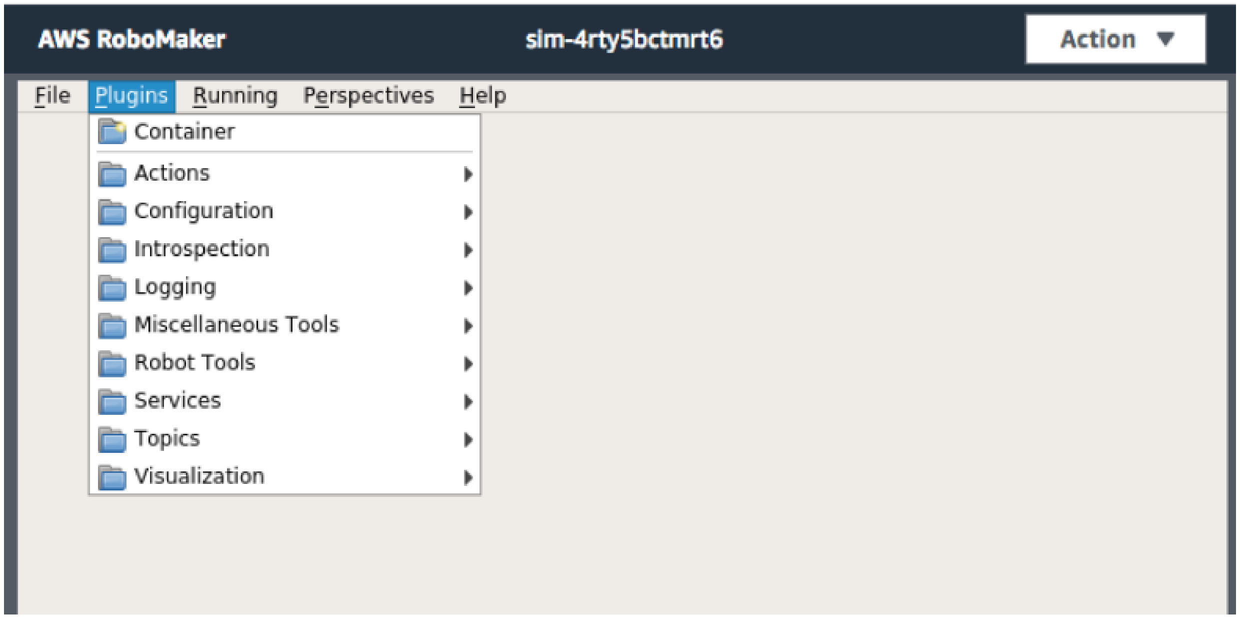
a. To view vehicle learning in 3D simulations, double-click at the “Gazebo”.



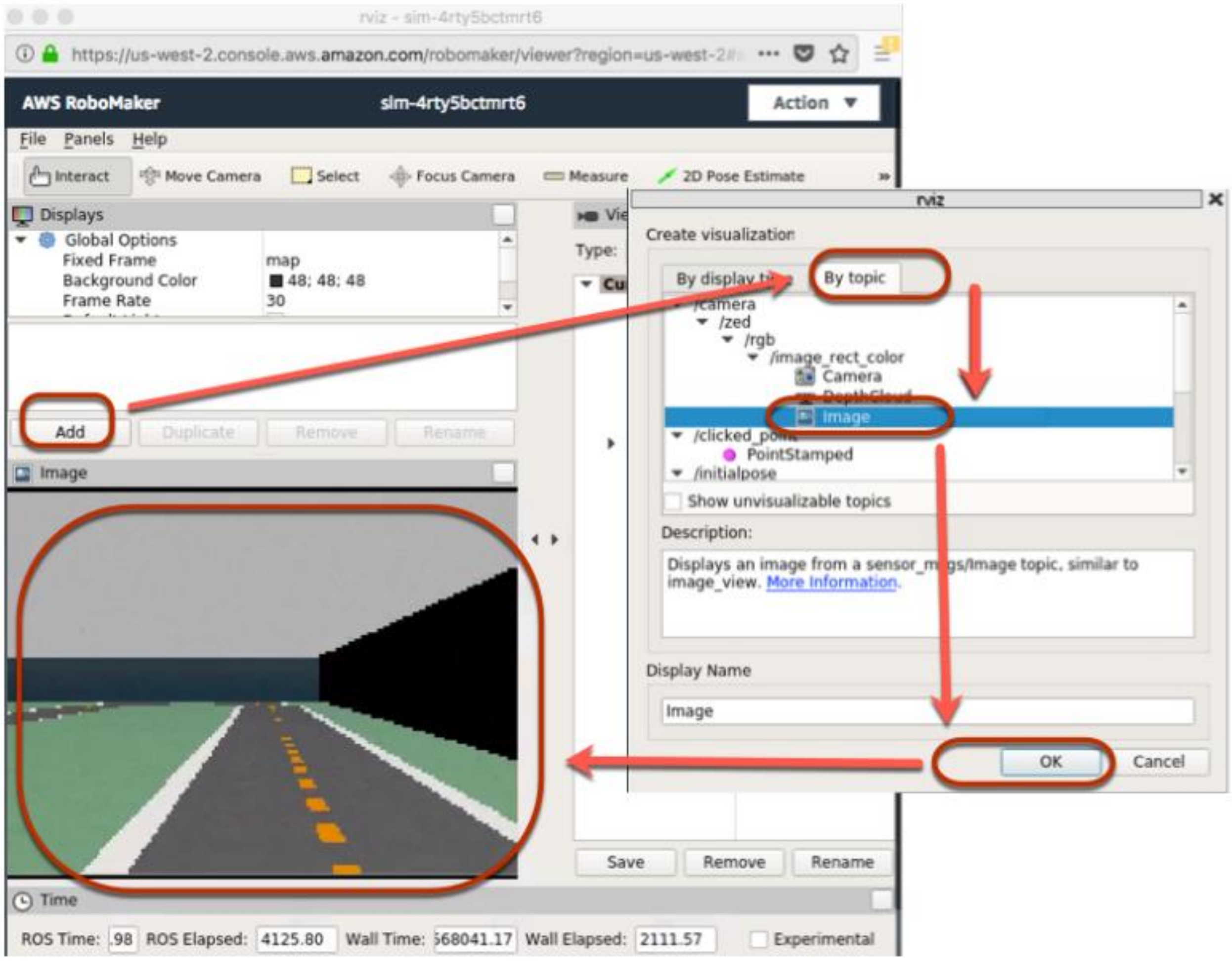
Property	Value
name	racetrack
is_static	✓ True
self_collide	False
pose	
link	racetrack::link

Real Time Factor: 1.95 Sim Time: 00 00:08:45.162 Real Time: 00 00:04:28.543 Iterations: 52

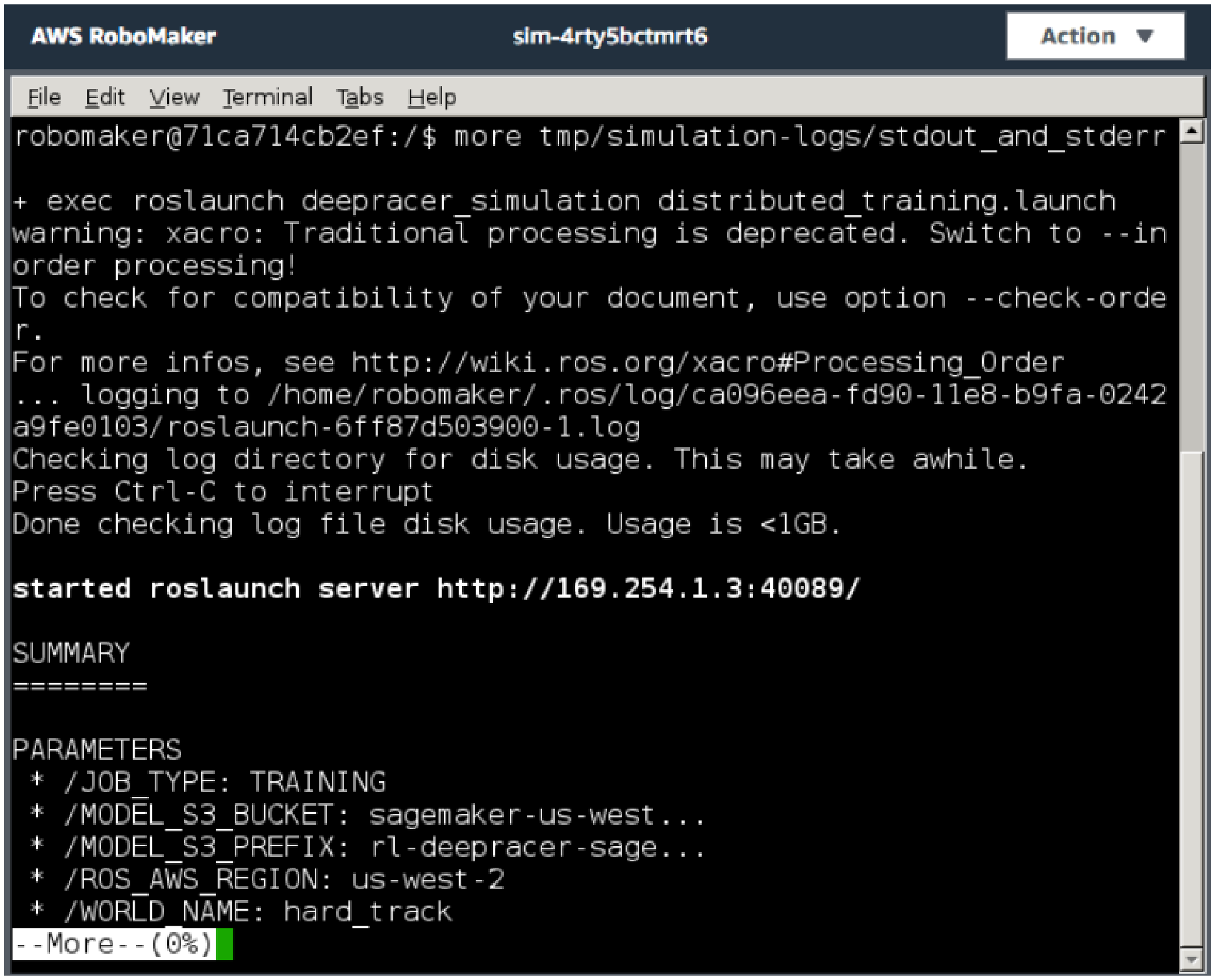
b. Double click on “rqt” to access the “rqt” utilities and select plug-in.



c. To see the front-facing vision of vehicle, tap at "rvis". Click on Add > By Topic and scroll down to "/camera/zed/rgb/ image_rec_color/Image" and click "OK".



d. To open a terminal window on simulation host, click on the "Terminal" and write suitable shell command.



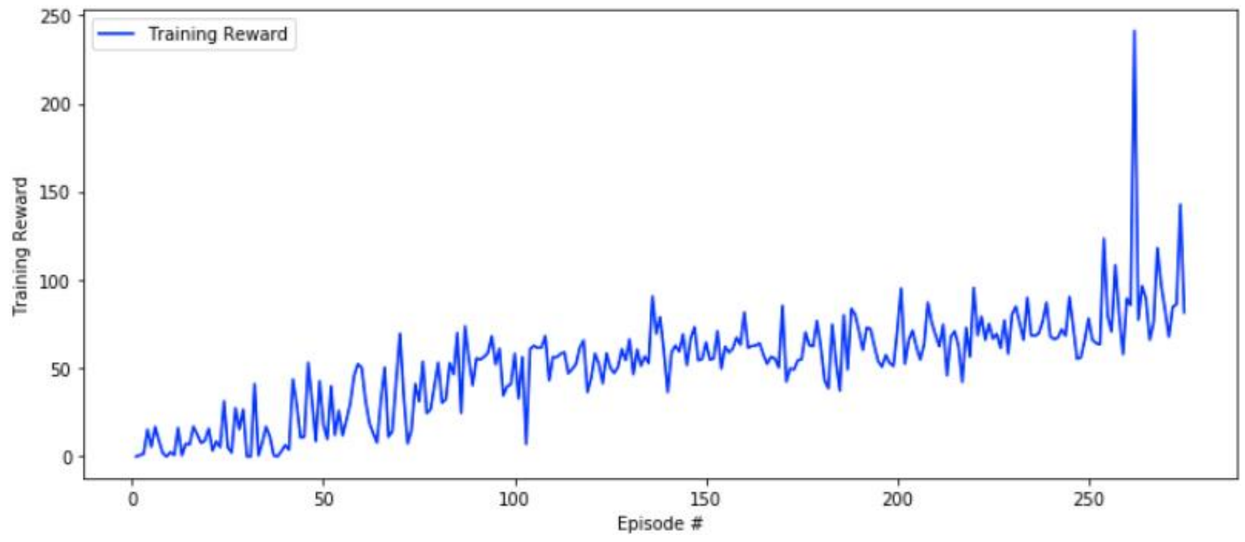
```
robomaker@71ca714cb2ef:/$ more tmp/simulation-logs/stdout_and_stderr

+ exec roslaunch deepracer_simulation distributed_training.launch
warning: xacro: Traditional processing is deprecated. Switch to --in
order processing!
To check for compatibility of your document, use option --check-orde
r.
For more infos, see http://wiki.ros.org/xacro#Processing_Order
... logging to /home/robomaker/.ros/log/ca096eea-fd90-11e8-b9fa-0242
a9fe0103/roslaunch-6ff87d503900-1.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://169.254.1.3:40089/

SUMMARY
========

PARAMETERS
 * /JOB_TYPE: TRAINING
 * /MODEL_S3_BUCKET: sagemaker-us-west...
 * /MODEL_S3_PREFIX: rl-deepracer-sage...
 * /ROS_AWS_REGION: us-west-2
 * /WORLD_NAME: hard_track
--More--(0%)
```

To see the rewards for last 10 steps, type the following command:

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

6. To envision the performance of the trained model in simulation, execute the two code cells under "Plot metrics for training job". On successful completion, following plot of "Training reward vs Episode #" appears.

When the user is done with model evaluation and wants to terminate simulation application then run the code of "Clean Up Simulation Application Resource".



6 Operating AWS Deep Racer Vehicle

After completion of training and evaluating AWS DeepRacer model in the AWS DeepRacer simulator, user can deploy the model in AWS DeepRacer vehicle. In this ways user can assess the performance of the model in physical environment which mimics the real-world Autonomous race.

Following list must be checked before driving vehicle for the first time.

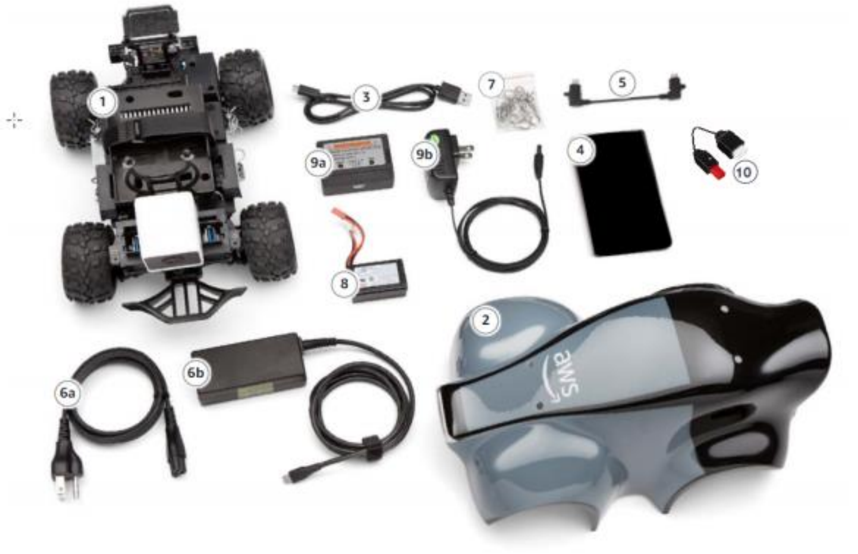
- Setup the vehicle as shown in above steps
- Install the updated software
- Calibrate drive-chain-sub-system.

6.1 Get Started with AWS Deep Racer Vehicle

AWS DeepRacer vehicle is machine learning-enabled, Wi-Fi supported 1/18th scale model mounted with 4-megapixel camera at the front. Its computation model is based on Ubuntu-Linux. By running different inference based on the reinforcement learning model the vehicle can drive itself.

As about its component, this vehicle is powered by brushed motor. To control the driving speed, it is equipped with a voltage regulator that controls the speed of motor spin. All the components are protected in a black cover premium quality AWS vehicle chassis.

6.1.1 Inspecting AWS DeepRacer Vehicle



Certain components can be found in the box for vehicle. These includes:

- Vehicle Chassis (1)
- Vehicle body shell (2)
- Micro-USB to USB-A cable (3)
- Compute battery (4)
- Compute battery connector cable (5)
- Power cable (6a)
- Power adapter (6b)
- Pins (spare parts) (7)
- Vehicle battery (8)
- Vehicle battery charge adapter (9a)
- Vehicle battery charge cable (9b)
- Battery unlock cable (10)

Before Setting up the AWS DeepRacer vehicle, following are pre-requisites that must be ready.

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

6.1.2 Charging and Installing AWS DeepRacer Batteries

AWS DeepRacer vehicle has two power sources.

- vehicle battery
- computing module power bank

The power bank runs the compute module. The compute module retains all the action AWS DeepRacer like sustaining the Wi-Fi connection, it also runs inferences deployed by user.

The vehicle battery gives motor the powers to move the vehicle. It comes with two sets of cable. These cable are differently colored and like red and black and white. One set is used to connect to the charger and other set of cable is for its connection to the vehicle's ESC.

After completely charged, the battery voltage will drop as the batteries discharge. When the voltage drops, the available torque also drops. As an outcome, a similar speed setting will result in more slow speed on the track. At the point when the battery is completely vacant, the vehicle stops moving. Under normal conditions, the battery usually lasts 15-25 minutes. The assurance of liable conduct, it is suggested that user should charge the battery after like clockwork of utilization.

Following are the steps to follow to install and charge the vehicle battery and the power bank.

1. Remove AWS DeepRacer vehicle shell.
2. Remove the four vehicle chassis pins. Carefully lift vehicle chassis while keeping wires connected.
3. To charge and install the vehicle battery, do the following:
   a. To charge the battery, plug the three-wired cable set from the batter to the charger to connect the battery to the power adapter and then plug the power adapter to a wall outlet or to a USB port if a USB cable is used to charge the battery
   b. After the battery is charged, plug the two-wired cable set of the vehicle battery cable into the black and red cable connector on the vehicle.
   c. 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.
4. To check if the vehicle battery is charged, do the following:
   a. Slide the vehicle power switch to turn on the vehicle.
   b. Listen for two short beeps. If user don't hear the beeps, the vehicle is not charged. Remove the battery from the vehicle and repeat Step 1 above to recharge the battery.
   c. When not using the vehicle, slide the vehicle power switch back to turn off the vehicle battery.
5. 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 changed. 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.
6. 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.

6.1.3 Testing AWS DeepRacer Compute Module

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

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, user can log in to the compute module after user 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**. User 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.
4. When 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.

6.1.4 Turning-off AWS DeepRacer Vehicle

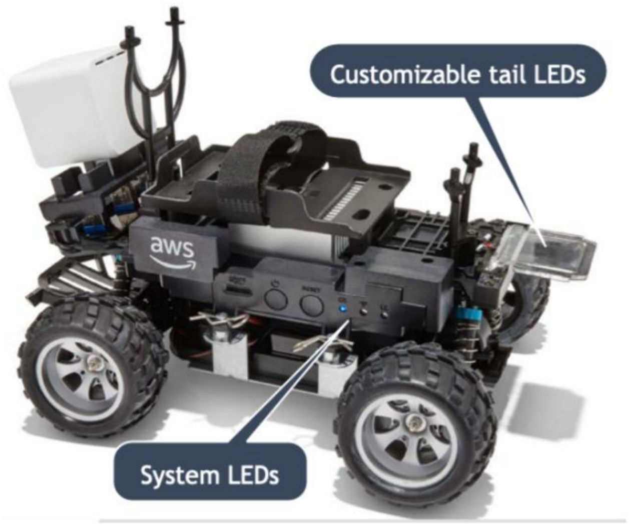
To turn off AWS DeepRacer vehicle, unplug the vehicle from the external power source. Pressing the power button on the device until power indicator is off.

6.1.5 AWS DeepRacer Vehicle LED Indicators

AWS DeepRacer vehicle has two sets of LED indicators for the vehicle status and for customizable visual identification of vehicle, respectively.

The AWS DeepRacer vehicle system LED indicators are located on the left side of the vehicle chassis when the vehicle is in the forward position in front of you. The three system LEDs are positioned after the “RESET” button. The first LED shows the status of the system power. The second (middle) LED is reserved for future use. The last (right) LED shows the status of the Wi-Fi connection.

The AWS DeepRacer vehicle custom LEDs are positioned at tail of vehicle. They're used to help identifying vehicle’s in races when multiple vehicles are present.



6.2 Setting Up the Vehicle

6.2.1 Getting Ready for Wi-Fi Connection

To make connection of computer with the vehicle's compute module, following set up is required.

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

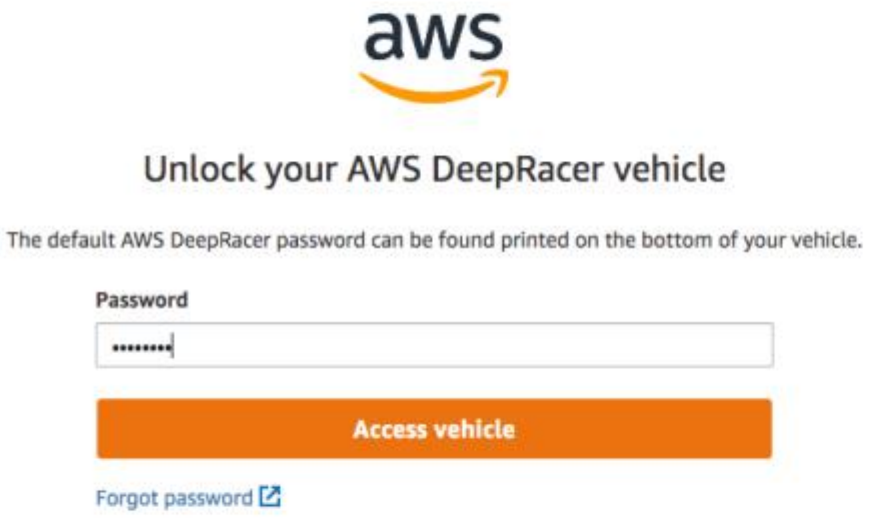
6.2.2 Setting Up the Wi-Fi connection and Updating Vehicle's Software

1. Beneath the vehicle, note down the password printed under "Host name" that is needed to log in to the console to perform setup.
2. On browser, go to "https://deepracer.aws" to launch device control console of vehicle.
3. Under "Unlock AWS DeepRacer vehicle", enter the password and click on "Access vehicle".
4. On "Connect vehicle to Wi-Fi network" pane, select Wi-Fi network name from "Wi-Fi network name (SSID)" and join the network.
5. On "Software update" pane, if a software update is required, turn on the vehicle's compute module, and choose "Install software update" and wait until the software update successfully.
6. The IP address shown under "Wi-Fi network details" will be required to open the vehicle's device control console.

6.3 Launching AWS DeepRacer Vehicle console

To access device console of the AWS DeepRacer vehicle via Wi-Fi connection, consider the following steps:

1. To get the device console of the vehicle, open web browser and write down the vehicle's IP address.
2. Under "Unlock AWS DeepRacer vehicle", write down the device console's password in "Password" and click on "Access vehicle".



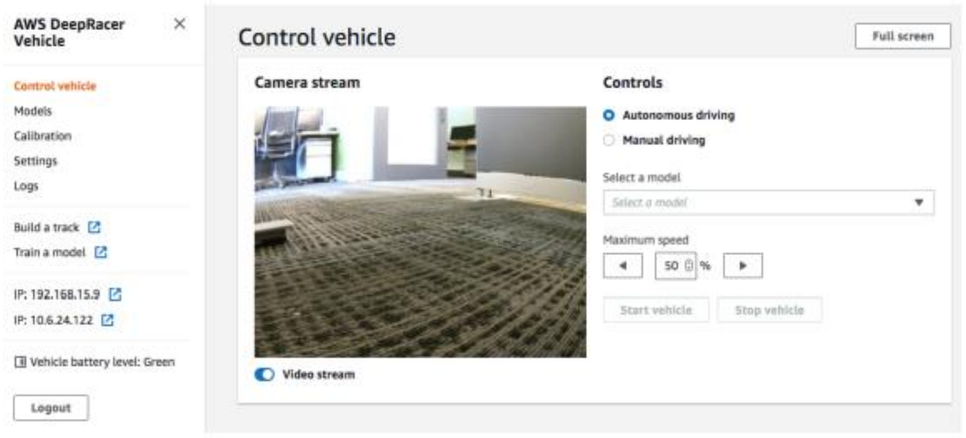
Unlock your AWS DeepRacer vehicle

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

Password

••••••••

3. On successfully signed in, following console's home page appears. Vehicle then becomes ready to calibrate and operate.


AWS DeepRacer Vehicle

Control vehicle
Models
Calibration
Settings
Logs

Build a track
Train a model

IP: 192.168.15.9
IP: 10.6.24.122

Vehicle battery level: Green

Logout

Control vehicle

Full screen

Camera stream

Video stream

Controls

Autonomous driving
Manual driving

Select a model: Select a model

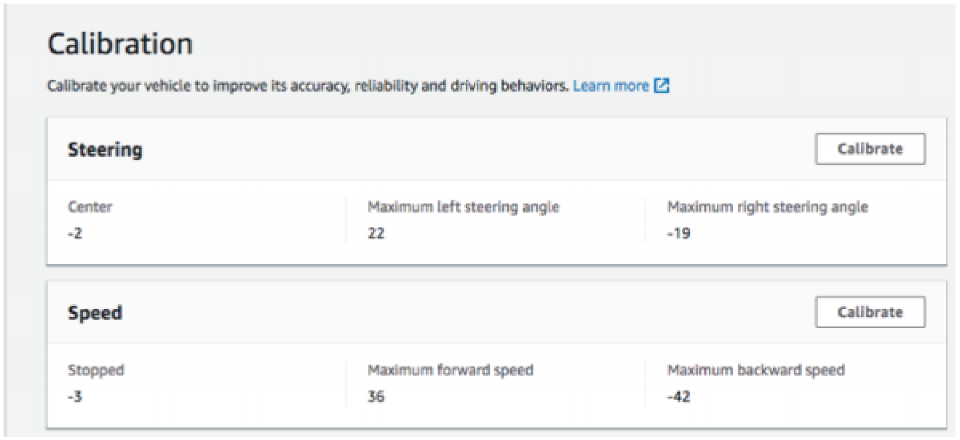
Maximum speed: 50 %

Start vehicle
Stop vehicle

6.4 Calibrating AWS DeepRacer Vehicle Model

To calibrate the vehicle, do the following steps:

1. Click on "Calibration" from the navigation pane.


Calibration

Calibrate your vehicle to improve its accuracy, reliability and driving behaviors. Learn more

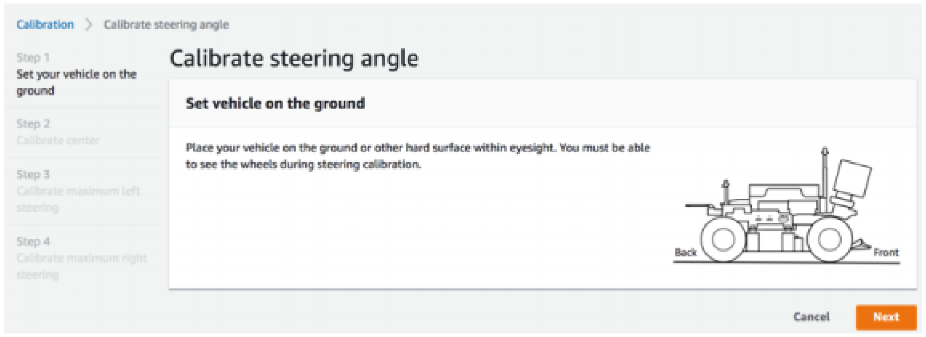
Steering — Calibrate

Center	Maximum left steering angle	Maximum right steering angle
-2	22	-19

Speed — Calibrate

Stopped	Maximum forward speed	Maximum backward speed
-3	36	-42

2. On the "Calibration" page, select "Calibrate" in "Steering" and consider the following steps below for vehicle's calibration.
   a. Place vehicle on the hard surface where wheels can be seen while steering calibration and press "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

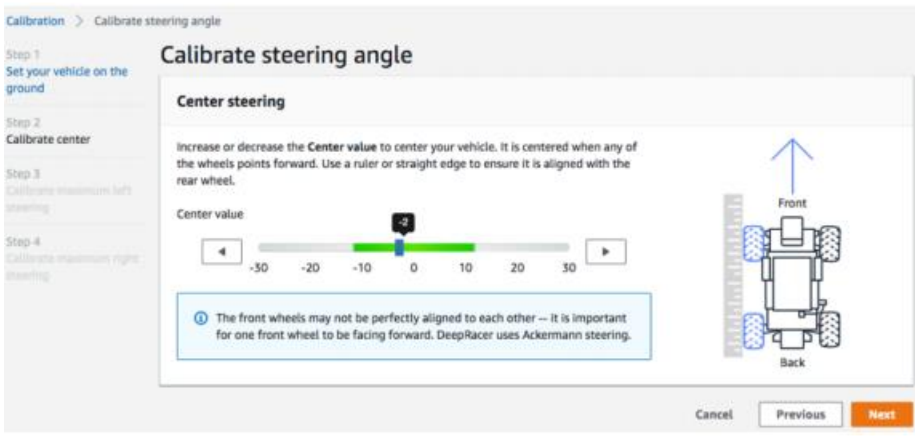
Calibrate steering angle

Set vehicle on the ground

Place your vehicle on the ground or other hard surface within eyesight. You must be able to see the wheels during steering calibration.

Cancel
Next

   b. Under "Center steering", slightly move the slider left or right to position where front wheels of vehicle are aligned with rear wheel on the same side and press 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

Center steering

Increase or decrease the **Center value** to center your vehicle. It is centered when any of the wheels points forward. Use a ruler or straight edge to ensure it is aligned with the rear wheel.

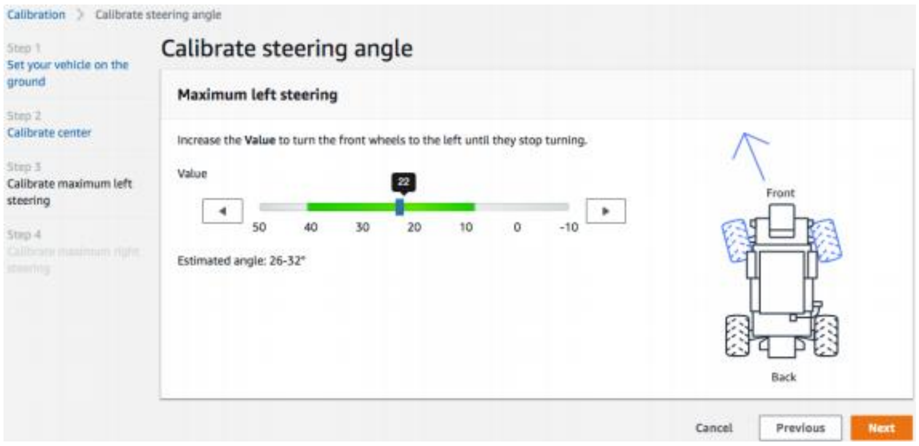
Center value

-2

-30 -20 -10 0 10 20 30

The front wheels may not be perfectly aligned to each other -- It is important for one front wheel to be facing forward. DeepRacer uses Ackermann steering.

c. Under "Maximum left steering", gradually move slider to the left until vehicle's front wheels halts turning left. This position agrees to be the "maximum left steering angle" and press 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.

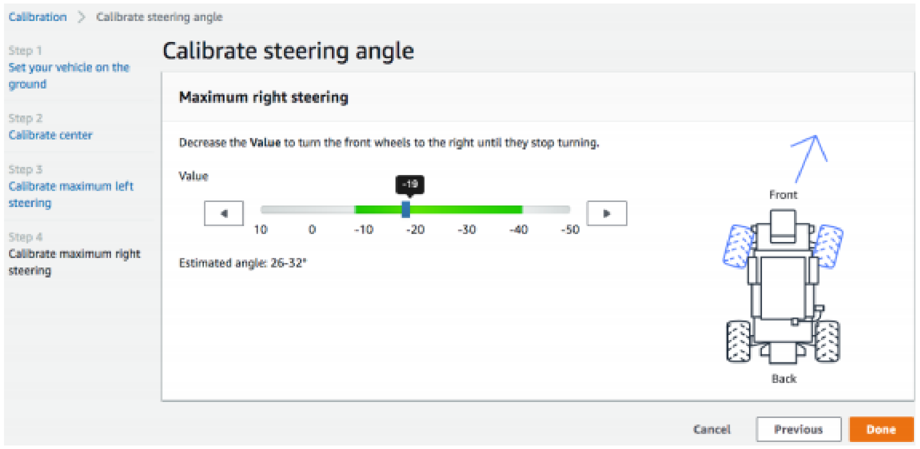
Value

22

50 40 30 20 10 0 -10

Estimated angle: 26-32°

d. Similarly repeat the point "c" for "maximum right steering" and press 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.

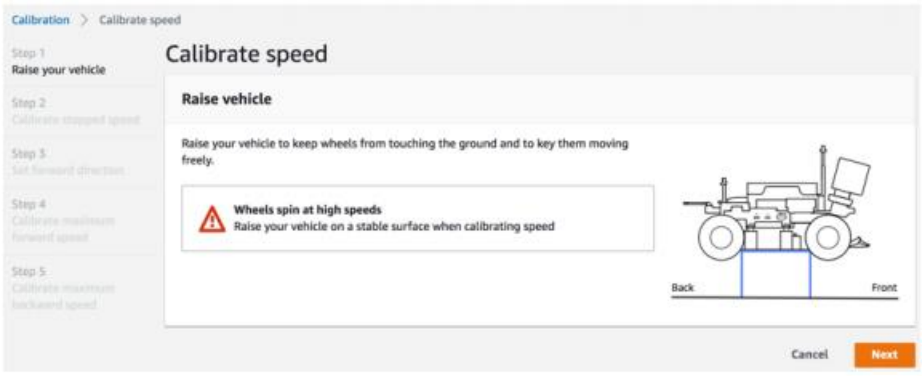
Value

-19

10 0 -10 -20 -30 -40 -50

Estimated angle: 26-32°

3. To calibrate maximum speed of vehicle, click on "Calibrate Speed" and follow the steps below.
   a. Increase the vehicle such that wheels are free to move left or right and click 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

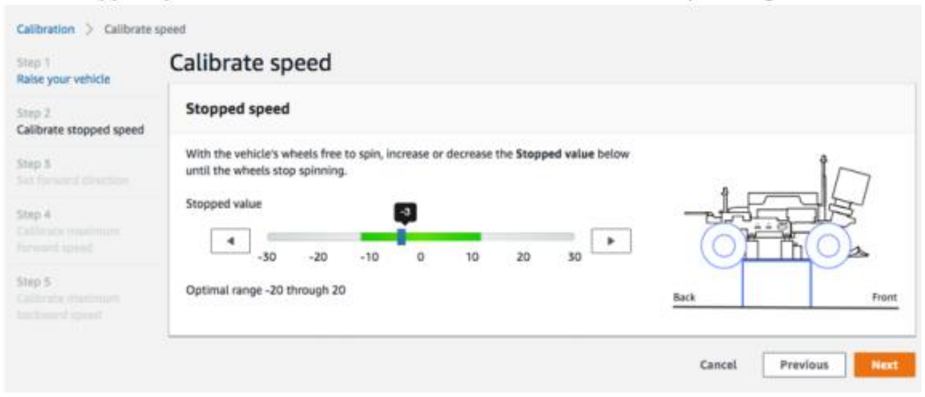
Calibrate speed

Raise vehicle

Raise your vehicle to keep wheels from touching the ground and to key them moving freely.

Wheels spin at high speeds
Raise your vehicle on a stable surface when calibrating speed

b. To calibrate stopped speed, click the left or right arrow to change "Stopped value" under "Stopped speed" until wheels stop turning and press 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

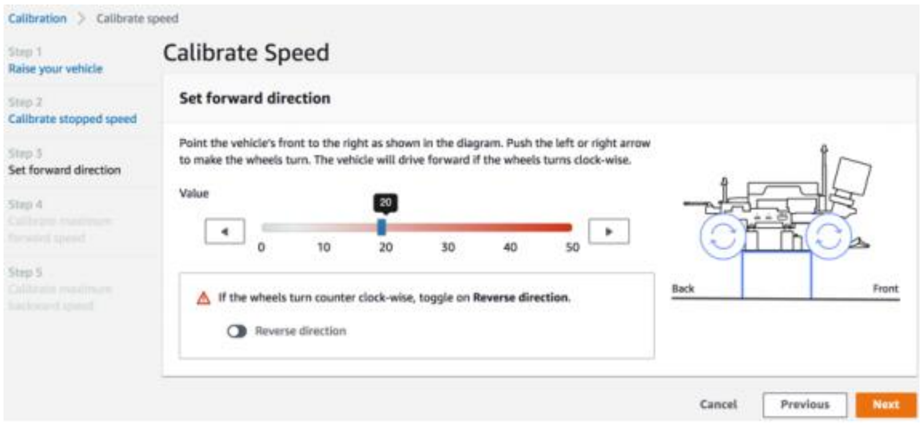
Stopped speed

With the vehicle's wheels free to spin, increase or decrease the **Stopped value** below until the wheels stop spinning.

Stopped value

Optimal range -20 through 20

c. To set forward direction of vehicle, place the vehicle as shown. Press the left or right arrow to make wheels turn. If they turn clock wise, forward direction is set. If not, toggle "Reverse direction" and click 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

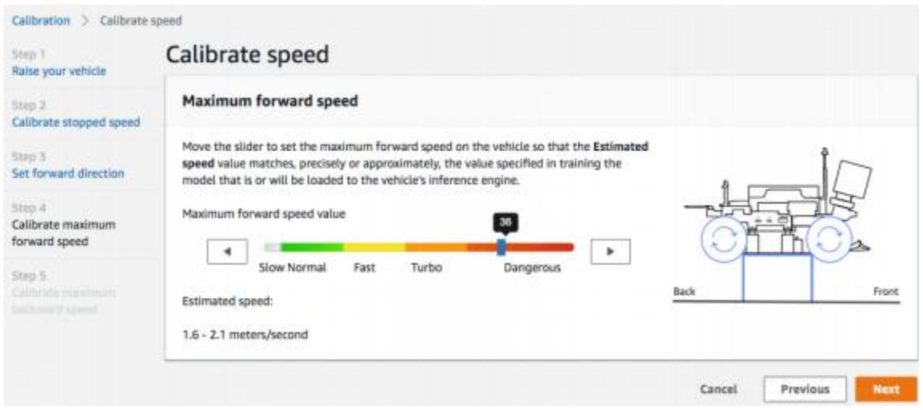
Point the vehicle's front to the right as shown in the diagram. Push the left or right arrow to make the wheels turn. The vehicle will drive forward if the wheels turns clock-wise.

Value

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

Reverse direction

d. To calibrate maximum forward speed, under "Maximum forward speed", move "slider" to fine-tune "Maximum forward speed value number" such that the "Estimated speed value" becomes equal to "maximum speed" stated in the simulation and press 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.

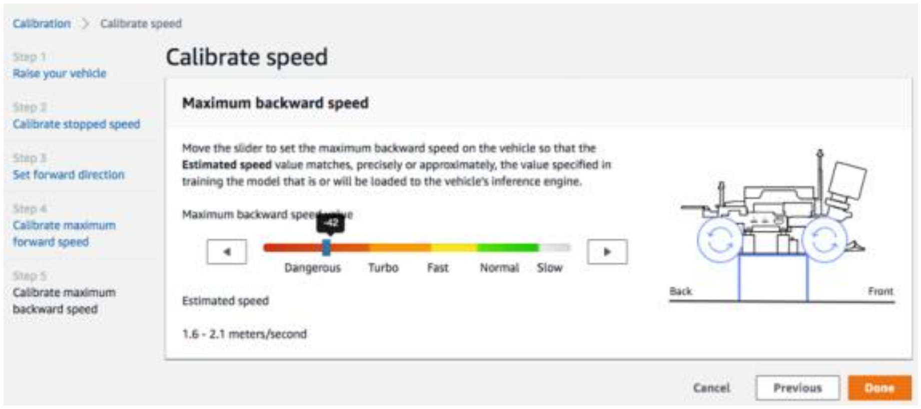
Maximum forward speed value: 36

Slow Normal Fast Turbo Dangerous

Estimated speed:

1.6 - 2.1 meters/second

e. Repeat the process as elaborated in point ‘d’ to calibrate “maximum backward speed”, and press Done.


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 backward speed

Move the slider to set the maximum backward 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 backward speed value: -42

Dangerous Turbo Fast Normal Slow

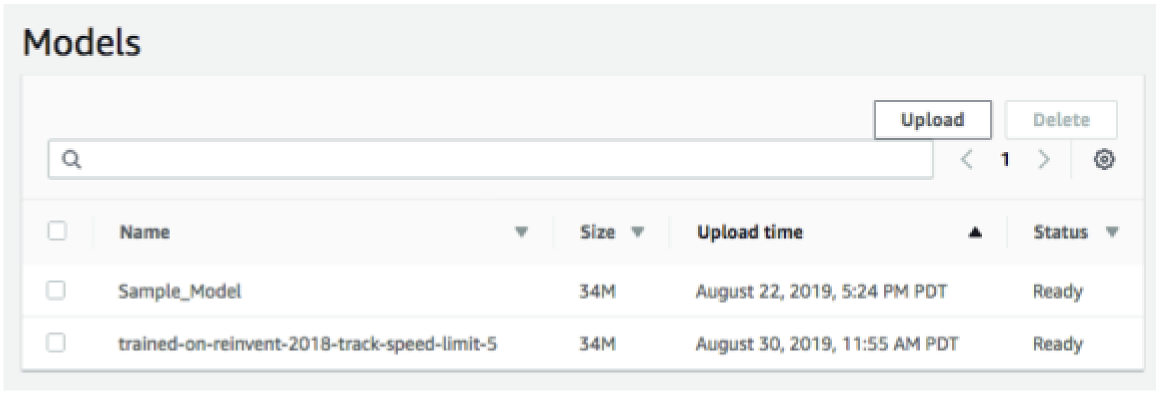
Estimated speed

1.6 - 2.1 meters/second

6.5 Uploading the Model

To upload the vehicle model, do the following:

1. Select “Model” from console’s navigation pane and click on “Upload”.


Models

	Name	Size	Upload time	Status
☐	Sample_Model	34M	August 22, 2019, 5:24 PM PDT	Ready
☐	trained-on-reinvent-2018-track-speed-limit-5	34M	August 30, 2019, 11:55 AM PDT	Ready

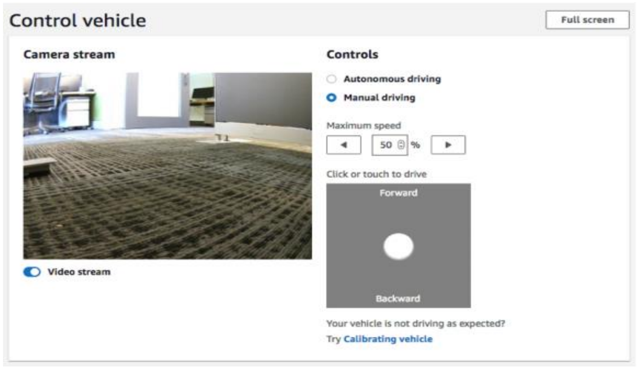
2. Pick the file from computer and Upload it.

6.6 Driving AWS DeepRacer Vehicle

AWS can be driven in two ways, i.e., manually and automatically. Details for each is illustrated here below:

6.6.1 Drive Vehicle Manually

1. After sign in, go on “Control vehicle page” and click “Manual driving”.



Control vehicle

Camera stream

Controls

Autonomous driving [✓ Manual driving]

Maximum speed 50 %

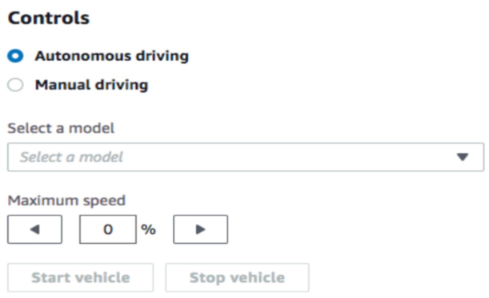
Click or touch to drive

Your vehicle is not driving as expected?
Try Calibrating vehicle

2. Under “Click or touch to drive”, point out the position within driving pad for vehicle to drive. Captured images of vehicle are displayed here.
3. Toggle the “Video stream” option for turning on or off the video.
4. Repeat these steps to move the vehicle on different locations.

6.6.2 Drive Vehicle Automatically

1. After sign in, go on “Control vehicle page” and click “Autonomous driving”.



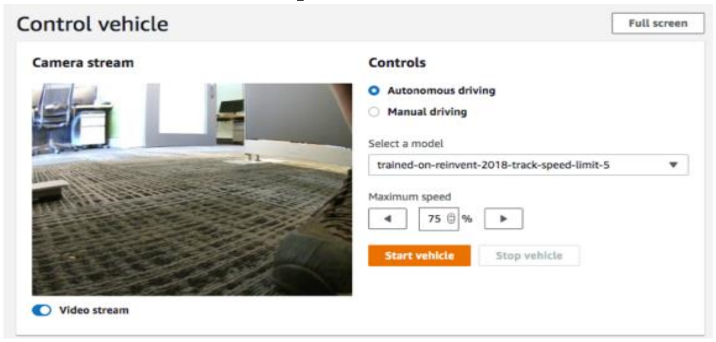
Controls

[✓ Autonomous driving] Manual driving

Select a model: Select a model

Maximum speed 0 %

2. From “Select a model” drop-down list, select the model to upload and click on “load Model”. It will 10 seconds to upload the model.
3. Adjust “Maximum speed” setting of vehicle as a percentage.
4. Click on “Start vehicle” for vehicle to drive automatically.
5. Turn the video on or off by toggling “Video Stream” option.
6. Lookout the vehicle while driving on physical track.
7. Click on “Stop Vehicle” if user wants to stop the vehicle.



Control vehicle

Camera stream

Controls

[✓ Autonomous driving] Manual driving

Select a model: trained-on-reinvent-2018-track-speed-limit-5

Maximum speed 75 %

7 Building the Physical Track for AWS DeepRacer

This section elaborates the steps to build physical environment where AWS vehicle can be deployed for analyzing its performance.

7.1 Track Materials and Building Tools

Before constructing the track, following material and tools are required to be ready beforehand.

7.1.1 Material Requirement

Following material is required to build the track.

1. For track borders
    - Tape of about 2-inches (white/ off-white for dark colored track)
2. For track surface
    - Hard floor e.g. hardwood, concrete, carpet, or asphalt felt (imitating real world road surface).
    - Interlocked foam
    - Rubber pads

7.1.2 Tools Requirement

Following tools are required to build the track for AWS DeepRacer vehicle.

1. Tape measure
2. Pair of scissors
3. Protractor
4. Ruler
5. Pencil
6. Knife
7. Compass

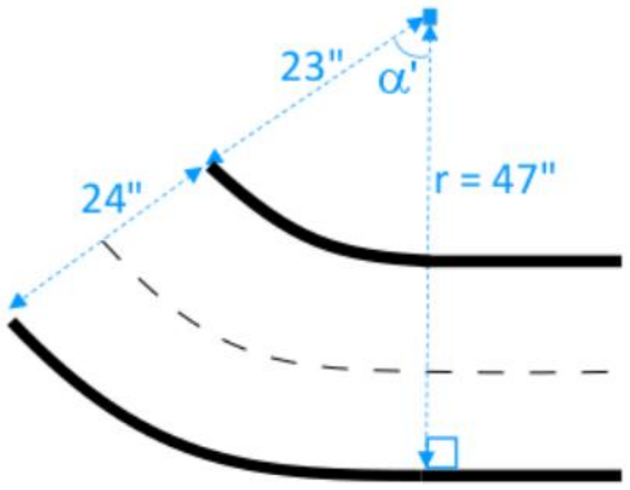
7.2 Building the Track for AWS DeepRacer Vehicle

For building the track for vehicle, it is recommended to always start with the straight path first and then gradually move towards the curvy path. It will help in increasing the vehicle's performance.

7.2.1 Dimensional Requirements

Following dimensional requirements must meet in order to build the track.

1. The smallest turning radius (rmin) of track turning angle (α) at the corner should obey the following limits.
    - Track's turning angle # = π ≤ 90 degrees, rmin ≥ 25 inches, 30 inches (recommended).
    - Track's turning angle # = π > 90 degrees, α, rmin ≥ 30 inches, 35 inches (recommended).
2. The track width (wtrack) should fulfill the following limit: wtrack ≥ 24 ± 3 inches.
3. The track surface should be smooth and have uniform dark color. The least encircling area should be 30 inches' x 60 inches.
4. User should encircle track with same-colored barriers with at least 2.5 feet length and 2 feet far from track at the points.



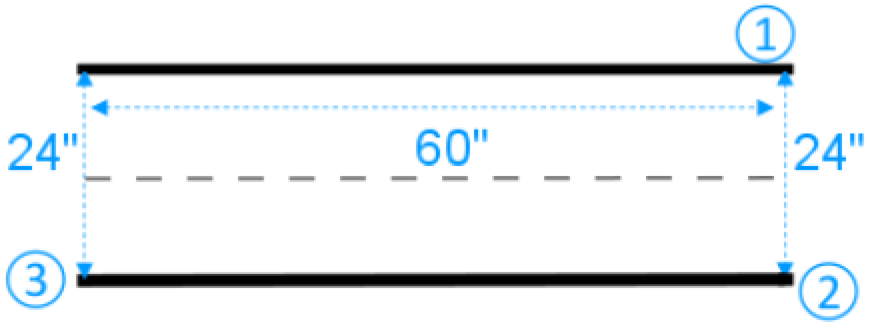
7.2.2 Consideration for Model Performance

Following factors must be considered while building the track.

1. Avoid placing white object near the track as training in simulated environment considers that only borders are white.
2. Always use clean and clear (continuous) tape for marking otherwise with broken tap, model performance can be degraded.
3. Do not use reflective surface for making the track floor.
4. Avoid using track floor with line markings other than track lines as model can interpret the non-track lines as a part of track.
5. Place obstacles nearby the track to reduce interferences from background objects.

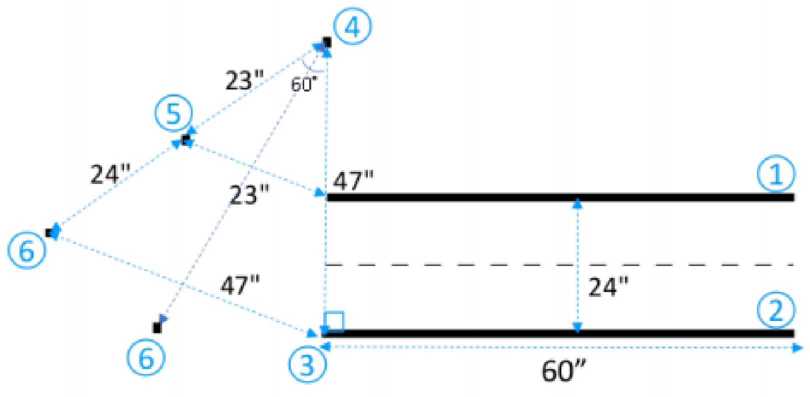
7.2.3 Steps to Build Track

1. To construct the straight path initially with one single turn can be made by attempting the following as illustrated.
   a. Place 60-inch long piece of tape on floor to lay down first border in a straight line (1).
   b. Use the tape measure to trace the second border's two end points (2) and (3).
   c. Place them all 24 inches apart from first border's two ends.
   d. Place another 60-inch long piece of tape on floor to lay down second boarder to connect the two endpoints (2) and (3).

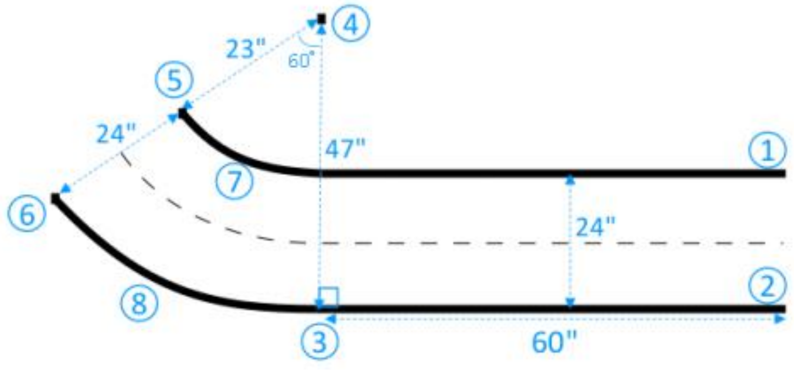


2. To turn the track at 60-degree angle, make the following illustrations:

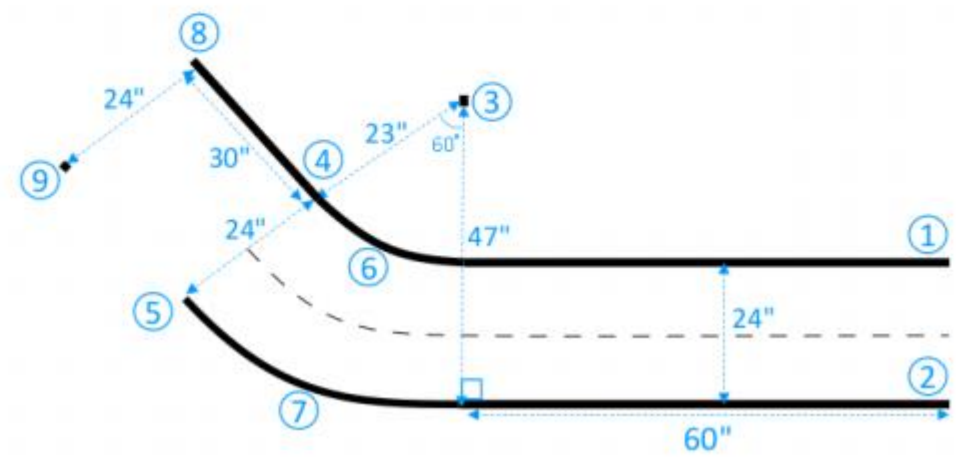
a. Use measuring tape to point out the center (4) of turning radius (4-3 or 4-6) and mark its center with piece of the tape.
b. Draw an equilateral triangle with three sides as shown by points (3-4), (4-6), and (6-3).



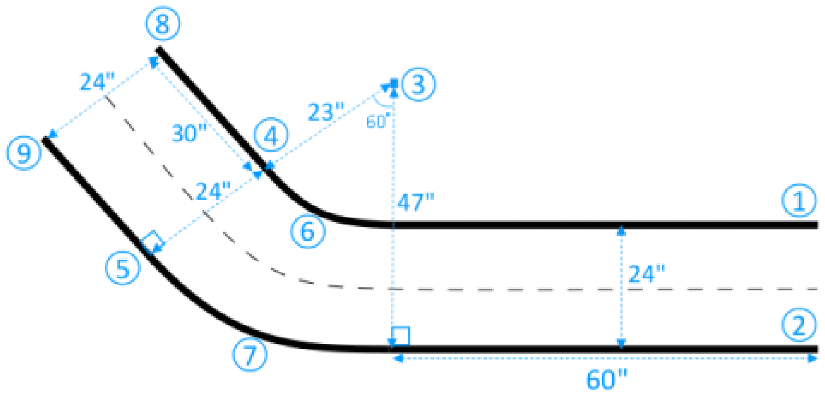
c. Place mini tape segments, such as 4-inches each, on floor to build the curved border (7) and (8) linking them with straight-line borders. These curved borders don't need to be parallel.



3. To prolong the track with next straight path of about 30 inches long and 24 inches wide, follow these steps:
   a. Place 30-inch long piece of tape on floor to make the first border (4-8) perpendicular to edge (3-5).



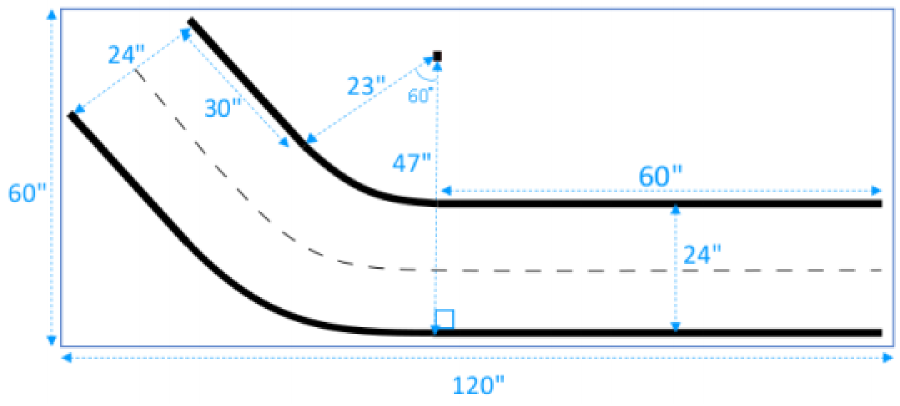
b. Use the measuring tape to find out the ending point of second border (9).
c. Place another 30-inch long piece of tape on floor to create the second border (5-9) perpendicular to the edge (3-5).



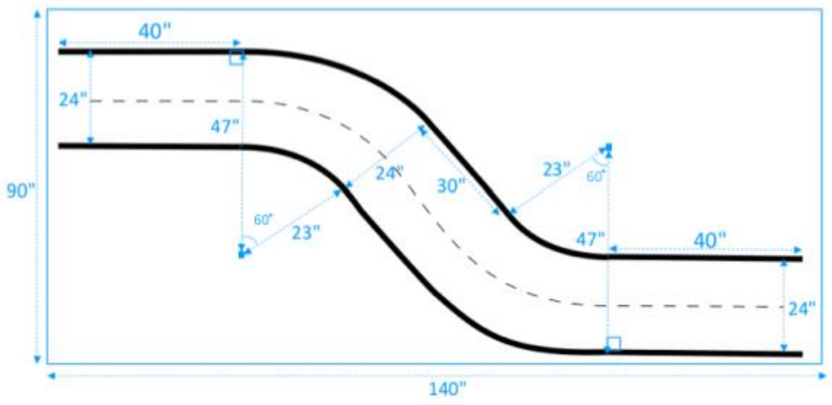
These were the steps that were needed to be followed to build the physical track for AWS DeepRacer vehicle.

7.3 Design Templates for AWS DeepRacer Track to Deploy Vehicle

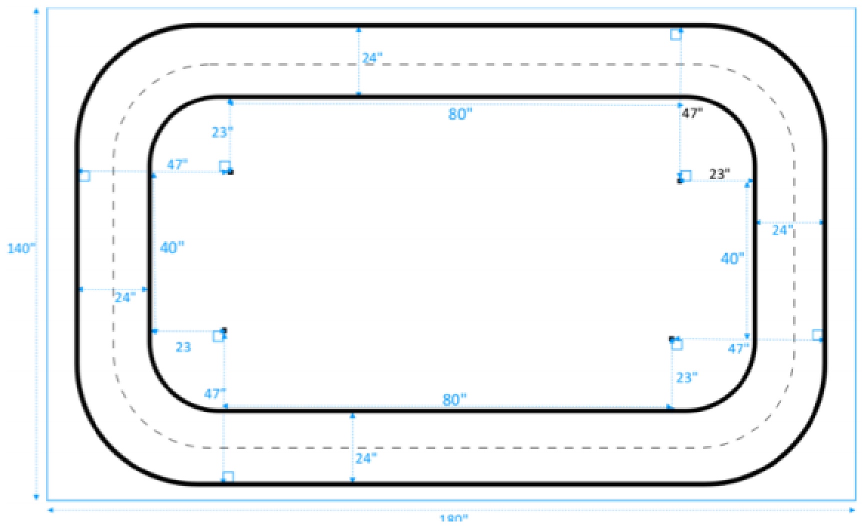
1. Template for Single-turn Track



2. Template for S-curved Track



3. Template for Looped Track


24"
80"
47"
23"
47"
23"
24"
40"
40"
24"
140"
23
47"
47"
23"
80"
24"
180"