

# AWS DeepRacer Simulation Report

Do-Gon Kim<sup>1</sup>, Shaokun Feng<sup>2</sup>

<sup>1</sup>Department of Mechanical Engineering, Columbia University, New York, United States

<sup>2</sup>Department of Electrical Engineering, Columbia University, New York, United States

*Abstract* — This report showcases the process of developing a software simulation system for AWS DeepRacer. It uses a Robot Operating System (ROS) and PyBullet physics engine to create a simulated environment for self-driving car research. We have built a strong simulation framework by combining ROS with Pybullet that enables detailed testing and fine-tuning of vehicle and environment models without physical hardware restrictions. The model of the AWS DeepRacer in Universal Robot Description Format (URDF) includes its chassis, wheels, cameras, LIDARs and other sensors necessary to accurately simulate vehicle dynamics and sensor feedback.

In our project, users can control the DeepRacer through keyboard inputs which are then translated into movement commands within the virtual world allowing for interactive real-time testing. We discuss technical difficulties faced, solutions employed as well as showcase system capabilities across different simulation scenarios. This not only strengthens research opportunities in autonomous vehicles but also sets foundations for future developments in simulation-based autonomous driving technologies.

*Keywords*—ROS, PyBullet, DeepRacer, URDF, Simulation

## I. INTRODUCTION

AWS DeepRacer represents an excellent educational platform where ML meets practical robotics applications in the field of self-driving cars. What makes our work unique is that while other groups focus on developing applications with it or even creating new ones entirely from scratch; ours will help build a software platform for simulating and controlling these vehicles virtually.

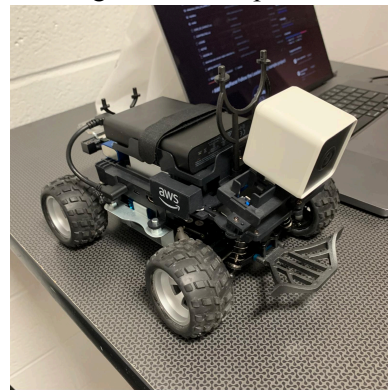
This article highlights various approaches used by our team towards designing software controls systems applicable across different AWS DeepRacer simulations using Robot Operating System (ROS). Our main aim was to enable more complex simulation

scenarios which allow users to test out their algorithms under unlimited hardware limitations.

To achieve this goal, we seamlessly integrated ROS with PyBullet thus bridging realistic physics together with simulated environments thereby making them easily accessible for research purposes where development can be done in a more controlled manner. The report talks about the challenges faced technically and how they were overcome alongside potential implications that could arise from such work on future autonomous vehicle simulations.

Through our project, we seek to empower all those who are passionate about AWS DeepRacer by giving them tools which until now seemed impossible. With the findings made it became clear that a simulated control system not only aids learning but acts as an essential step towards advanced autonomous driving technology. The source code of the project is available at [https://github.com/7Gao/DeepRacer\\_Simulator](https://github.com/7Gao/DeepRacer_Simulator) on the updated\_4\_29 branch.

Fig1. AWS DeepRacer



## II. SIMULATION PLATFORM

Throughout the process of creating simulation software for AWS DeepRacer, we experimented with a few different platforms: MuJoCo, PyBullet and Isaac Gym. Each platform has its strengths and weaknesses

depending on what kind of robotics or application one wants to achieve.

- MuJoCo is popularly known for simulating fast yet accurate dynamic systems involving robots as well as living organisms. This comes in handy when dealing with contact dynamics including friction that require precise physical interactions during simulations.[1]

- Isaac Gym is an NVIDIA powered environment designed specifically for large scale complex simulations accelerated by GPUs. It allows multiple agents being trained simultaneously within scenes therefore saving time due to efficient parallel computation of many similar processes at once.[2]

- PyBullet is a physics engine that simulates rigid body dynamics. It is free and open-source, and used widely in industry as well as academic research due to its easy integration, strong community support, and compatibility with Python.[3]

Table 1. The comparison of 3D platforms.

Feature	MuJoCo	PyBullet	Isaac Gym
Cost	Paid, with a free trial	Free	Free
Physics Accuracy	High accuracy and stability	High accuracy	High accuracy
Ease of Integration	Moderate	Easy, with Python support	Moderate, requires NVIDIA GPU
Support for Robotics	Extensive	Extensive	Extensive
Parallel Simulation	Limited	Good	Excellent, GPU-accelerated
Community and Support	Good, but smaller community	Very active and large community	Growing, backed by NVIDIA
Documentation	Well-documented	Extensively documented	Well-documented
Application Focus	Academic, research	Academic, research, hobbyist	Large-scale machine learning
Hardware Requirements	Moderate	Low	High (GPU intensive)
Real-time Simulation	Good	Good	Excellent

We decided to use PyBullet for several reasons. Firstly, it doesn't cost anything. Secondly, it has extensive physics simulation capabilities. Thirdly, it works seamlessly with the Robot Operating System (ROS). Additionally, the active development community of PyBullet and abundant documentation were invaluable to us because they helped in

troubleshooting issues encountered during the development process and also inspired new ideas.

### III. PYBULLET

Our simulation environment was created with great attention to detail so that AWS DeepRacer models can be tested under realistic conditions. In order to set up the environment we need various Universal Robot Description Format (URDF) files which define how objects behave physically within our simulation.

#### A. 3D Environment Setup

1. Base Environment Setup: We initialize the PyBullet physics engine and set gravity among other environmental parameters. Then we load a basic plane from URDFs to serve as ground:

```
1 p.setGravity(0, 0, -10)
2 p.loadURDF("plane.urdf")
```

2. Loading the AWS DeepRacer Model: We load an AWS DeepRacer model URDF that contains all information about the vehicle like where its camera or LIDAR sensor are placed:

```
1 deepracer_id = load_urdf("
    /deepracer_with_camera_and_lidar.urdf",
2 [4.5, 0, 0.1], [0, 0, -(1/2)*np.pi])
```

3. Adding Static Objects: To make the simulation more realistic we include roads, trees etc., which are positioned and rotated so that they create a challenging track for the DeepRacer:

```
1 road_from_Qi_gao = load_static_urdf("/road.urdf", [0,0,0],
    [np.pi/2,0,0])
2 tree1 = load_static_urdf("/tree.urdf", [-1,0.6,0],
3 [np.pi/2, 0, 0])
```

#### B. Car URDF Model

The AWS DeepRacer URDF (Universal Robot Description Format) file is an XML configuration file which describes the physical joints, links and other attributes of the robot. It includes everything from wheels and chassis to cameras and LIDAR sensors, so that it can be accurately simulated in environments such as Gazebo or PyBullet. The main components of AWS DeepRacer are:

## 1. Chassis and Wheels:

- The chassis link represents the main body of the vehicle, with properties like mass and inertia specified for realistic physics simulation.

- Each wheel is treated as a separate link (left\_rear\_wheel, right\_rear\_wheel, etc.) connected to the chassis through joints that dictate their possible movements; e.g., continuous type joints allowing rotation for rear wheels to mimic rolling motion.

## 2. Steering Mechanism:

- Steering mechanism is represented by left\_steering\_hinge and right\_steering\_hinge links which act as pivot points enabling the front wheels to turn thus control direction of the vehicle.

- Continuous rotation is also allowed by joints connecting steering hinges with front wheels (left\_front\_wheel\_joint and right\_front\_wheel\_joint) thus simulating wheel spin.

## 3. Sensors:

- A camera and a LIDAR sensor are defined in URDF. camera\_link and laser links are used to position these sensors on the vehicle correctly.

- Fixed joints attach sensors to the main body (zed\_camera\_joint and hokuyo\_joint), making sure they remain stationary relative to moving parts of the vehicle.

### C. AWS DeepRacer Controller

1. Velocity Control for Driving Motors: To move forward or backward, we set velocity for rear wheel motors using setJointMotorControl2 function. Throttle value is scaled appropriately based on simulation:

```
1 p.setJointMotorControl2(self.deepracer_id,  
2 self.wheel_joints['left_rear_wheel_joint'],  
3 p.VELOCITY_CONTROL,  
4 targetVelocity=throttle_value)
```

2. Position Control for Steering: Change in orientation of front wheel hinges leads to steering control. Desired steering angle (in degrees) is converted into radians for the simulation by the function which uses position control mode:

```
1 p.setJointMotorControl2(self.deepracer_id,  
2 self.wheel_joints['left_steering_hinge_joint'],  
3 p.POSITION_CONTROL,  
4 targetPosition=steering_angle_radians)
```

### D. Simulating IMU and LIDAR Sensors

1. 1. Inertial Measurement Unit (IMU): This data is obtained directly from PyBullet, where the current state of the vehicle provides it. The orientation in quaternion format and angular velocity are retrieved as follows:

```
1 orientation_quat, _ = p.getBasePositionAndOrientation(deepracer_id)  
2 _, angular_velocity = p.getBaseVelocity(deepracer_id)
```

2. Light Detection and Ranging (LIDAR): A real LIDAR sensor is approximated by casting multiple rays around a 360-degree pattern using PyBullet's ray tracing capabilities. The intersection of these rays with any objects in the environment is efficiently computed using rayTestBatch function:

```
1 ray_results = p.rayTestBatch(start_points.tolist(), end_points.tolist())  
2 lidar_data = [result[2] * lidar_range for result in ray_results]
```

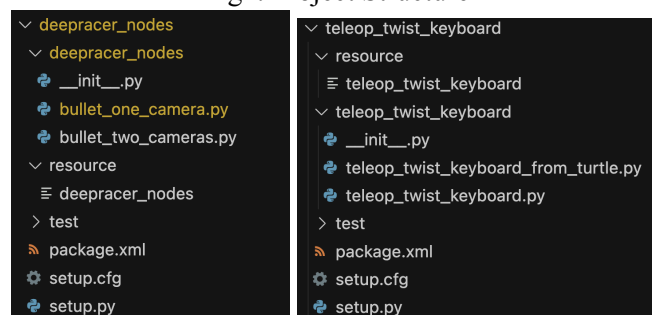
3. Camera: To simulate camera, a virtual one is set up that can capture images from the simulation environment PyBullet provides getCameraImage function, which takes into account position and orientation of camera to determine what it 'sees':

```
1 camera_img_data = p.getCameraImage(img_w, img_h,  
2 viewMatrix=view_matrix, projectionMatrix=projection_matrix,  
3 shadow=False, renderer=p.ER_BULLET_HARDWARE_OPENGL)
```

## IV. ROS

In our AWS DeepRacer simulation project, we use Robot Operating System (ROS) to interface between user inputs and vehicle control. The project is structured based on ROS framework as shown in Fig2:

Fig2. Project Structure



Speed control involves capturing keyboard inputs which are then published as messages on a ROS topic. These messages are received by the simulation node and used to instruct how throttle and steering angles should be adjusted for the vehicle.

#### A. Servo Message

*ServoCtrlMsg* is a custom ROS message defined as follows:

```
1 float32 angle
2 float32 throttle
```

This message helps structure the transmission of control signals over ROS. The keyboard node publishes *ServoCtrlMsg* messages to the topic `cmd_servo_control` using a publisher while the simulation node subscribes to the same topic receiving *ServoCtrlMsg* messages with a subscriber.

#### B. From Keyboard Input to Vehicle Control

1. Capturing Keyboard Inputs: A Python script is used to capture keyboard inputs by listening for key presses. Each vehicle movement or speed adjustment is associated with a specific key press. For example, pressing 'i' might mean move forward and 'o' might be interpreted as moving forward while turning right.

2. Mapping Keys to Movement Commands: Dictionaries (`moveBindings` and `speedBindings`) are used to map keys to movement commands. These bindings determine how each key affects the vehicle's speed and steering:

```
1 moveBindings = {
2     'i': (1, 0, 0, 0), # Move forward
3     'o': (1, 0, 0, -1), # Move forward and turn right
4     # Additional bindings...
5 }
```

3. Creating *ServoCtrlMsg* Messages: Whenever a key is pressed, it leads to creation of a new *ServoCtrlMsg* message. This message has two main fields which are;

- throttle: Controls the vehicle's forward or backward movement.
- angle: Controls the vehicle's steering angle.

4. Publishing Messages to the ROS Topic: After this, the *ServoCtrlMsg* gets published on a particular ROS topic called `cmd_servo_control`.

5. Receiving Messages in Simulation: In this case, within the simulation, node listens for new messages on `cmd_servo_control` topic where if any arrives then triggers callback function that processes:

```
1 def listener_callback(self, msg):
2     throttle_value = msg.throttle * 15 # Scale throttle for simulation
3     steering_angle = msg.angle * 15 # Scale steering angle for simulation
4     self.control_deepracer(throttle_value, steering_angle)
```

6. Controlling the Vehicle: Through PyBullet API, callback function receives and handles *ServoCtrlMsg* thus changing behavior of vehicle according to received data by setting wheel joint velocities (throttle) and steering hinge joint positions (steering)

## V. RESULT

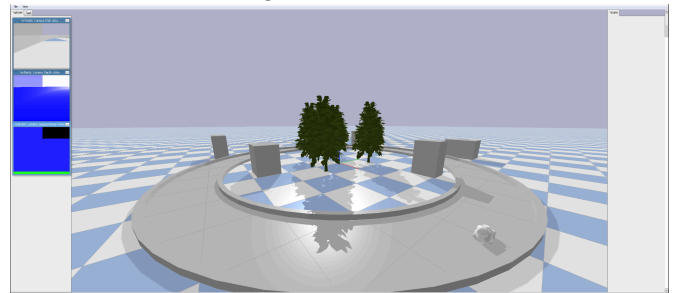
To highlight AWS DeepRacer simulation outcomes and ROS control via them we will take into account results obtained during simulations execution, teleoperating vehicles and monitoring ROS topics.

1. Running Simulations: To run simulation of AWS DeepRacer with one camera in PyBullet environment use the following command:

```
1 ros2 run deepracer_nodes bullet_one_camera
```

Once executed, this command initializes PyBullet simulation environment where AWS DeepRacer model along with its physical properties and camera are rendered. In simulation window you can see virtual environment in which vehicle moves. It also allows you to control it via teleoperation commands or predefined automation scripts.

Fig3. Simulation



2. Teleoperating the Deepracer: For real-time control using keyboard inputs, the command to run is:

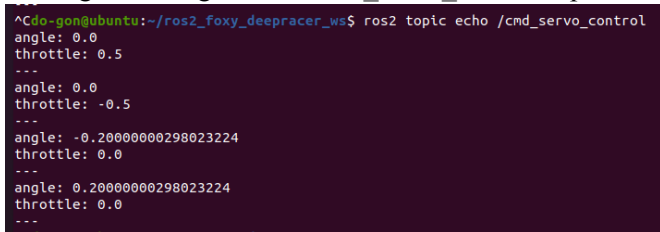
```
1 ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

This command will activate a node that captures keyboard inputs and converts them into ROS messages that control the movement of the vehicle. In other words, when keys are pressed, movements such as forward, backward, turn left or right are reflected on the simulation thereby enabling interactive control over an AWS DeepRacer.

3. ROS Topics Watch: This command helps to monitor in real-time what data is passing through ROS network hence giving an idea about which control signals have been sent to the vehicle and what sensor data it is producing.

## 1 ros2 topic echo cmd\_servo\_control

Fig4. Messages From *cmd\_servo\_control* topic



```
^Cdo-gon@ubuntu:~/ros2_foxy_deepracer_ws$ ros2 topic echo /cmd_servo_control
angle: 0.0
throttle: 0.5
---
angle: 0.0
throttle: -0.5
---
angle: -0.20000000298023224
throttle: 0.0
---
angle: 0.20000000298023224
throttle: 0.0
---
```

## VI. CONCLUSION

In conclusion, this integration has shown how feasible and effective simulations can be used for research and development on autonomous cars using AWS DeepRacer together with PyBullet within the ROS environment. Our work provides a flexible testbed where controlling algorithms can be tested and refined without relying on physical prototypes alone; thus overcoming their limitations. The results reported herein also highlight simulation speed up self-driving technology innovation process while making it more robust in different applications that may arise later in time.

## VII. REFERENCES

[1] Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). doi:10.1109/IROS.2012.6386109

[1] NVIDIA Corporation. (2021). Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. Retrieved from <https://developer.nvidia.com/isaac-gym>

[3] Coumans, E., & Bai, Y. (2016–2021). PyBullet, a Python module for physics simulation for games, robotics and machine learning. Retrieved from <http://pybullet.org>