Abstract

Traditional academic competitions that foster collaboration from student communities all around the globe have either been postponed indefinitely or canceled due to restrictions posed upon in-person gathering. Owing to this, virtual competitions are gaining importance as they provide the student community with an outlet for exposure and academic growth whilst having the convenience of being online and safe. Mars Society South Asia (MSSA) has developed a software stack capable of replicating on-site competition tasks via ROS-based simulations. The developed software enables users to perform necessary simulations without them having to interact with ROS or Gazebo. Currently, the software has been set out to be used for the Virtual Mars Rover Challenge (VMRC), a competition that focuses on simulating the next generation of Mars rovers and thereby providing the student community with a substitute to their on-site competitions. The objective of this white paper is to explain the software’s architecture, installation, usage, maintenance, and ease of use which is the singular most important factor in software adoption.

Keywords: ROS, Whitepaper, OpenAI Gym , gym-gazebo.

1 Introduction

Reinforcement Learning is a focused area of machine learning where an agent interacts with its environment and learns to take a series of actions to maximize the notion of cumulative reward. To build upon recent progress in Reinforcement Learning, the research community needs good benchmarks on which to compare cutting-edge algorithms. OpenAI [1] aims to combine the best elements of these previous benchmark collections, in a software package that is maximally convenient and accessible. It is a standardized toolkit for comparing different Reinforcement Learning paradigms. The gym library is a collection of community-maintained test problems (environments) that have a shared interface allowing the user to test different algorithms. Unfortunately, gym does not facilitate the provision of interfacing with the Robot Operating System (ROS) [13]. Robot Operating System ROS is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Gazebo [7] is the default ROS simulator, while ROS serves as the interface for the robot. gym-gazebo [17] is a complex piece of software that essentially acts as the OpenAI extension for Gazebo based environments. The extension enables us to use gym functionalities on Gazebo-based environments which simplifies the simulation process considerably.

2 Background

There are two basic entities: the environment which is the world the robot is present in and the agent which is the robot itself. The Agent-Environment Interaction (AEI) model [3] is composed of three main elements: the Capability Model, the Environment Model, and a set of interactions between capabilities and environment objects. Essentially, agents possess capabilities that sense and act upon objects in the environment via interactions. The figure provided below depicts the integration of these three parts into

---

1 https://github.com/singhnishesh/Virtual-Mars-Rover-Challenge-2021.
the AEI Model. The top part of the figure represents the Capability Model, which defines capabilities as consisting of a set of actions, each of which has a single operation that interacts with environment objects. The bottom part of the figure captures the Environment Model, which includes an explicit environment that contains a set of environment objects (that includes agents) and their relationships. The environment objects are governed by processes that implement specific environmental principles. Interactions are defined by the intended effect of operations on environment objects.

![Diagram of AEI Model](image)

**Figure 1:** Agent-Environment Interaction (AEI) model [3].

Env is the core gym interface. To interact with the environment via the agent, gym provides 3 main API methods that users of this class need to know are:

- **step(self, action):** steps through the environment by one timestep and returns a `done` flag after completion.
- **reset(self):** resets the state of the environment.
- **render(self):** renders one frame of the agent-environment interaction.

**gym-gazebo** provides the same functionalities for a ROS-based environment with the addition of a `get_observation(self)` method that allows arbitrary repeated access to the current observation. The **gym.make(self, "environment_name")** method at its core calls the constructor corresponding to the registered environment. The pseudo-code below demonstrates how to iterate the agent through the environment for successive time steps. Furthermore, the code resets the environment to its initial state when a condition is satisfied.

```python
import gym
env = gym.make('environment_name-v0')
while not done:
    action = env.action_space.sample()
    env.step(action)
    if(env.get_observation() > threshold):
        done = True
        observation = env.reset()
```

The developed software stack is centered around the **gym-gazebo** package and its functionalities. It incorporates some new modifications from our end while simplifying the installation process greatly. Furthermore, the modifications currently strictly adhere to the class of mobile ground robots and more specifically differential drive robots. Additions also include the inclusion of the setup file, a bash script that interacts with the
user and upon successful execution automatically integrates a controller for differential drive wheel systems, gazebo plugins from Team Hector which currently contains a 6 wheeled differential drive plugin, an IMU sensor plugin, an earth magnetic field sensor plugin, a GPS sensor plugin and a sonar ranger plugin, a generic keyboard teleoperation package for twist robots, and other necessary sensor plugins with the preliminary ROS package of the robot provided by the software operator.

The user input to the software includes the robot link’s physical properties such as geometry, color, mass, and inertia, and the collision properties of the link which is encapsulated in the Universal Robot Description Format (URDF) file. To generate the URDF, the user is encouraged to use the Solidworks to URDF exporter, a SolidWorks add-in that allows for the convenient export of SolidWorks parts and assemblies into a URDF file. The exporter will create a ROS-like package that contains a directory for meshes, textures and robots (URDF files). For single SolidWorks parts, the part exporter will pull the material properties and create a single link in the URDF. For assemblies, the exporter will build the links and create a tree based on the SolidWorks assembly hierarchy. The exporter can automatically determine the proper joint type, joint transforms, and axes.

3 Architecture
The architecture consists of 3 independent software packages: OpenAI Gym, ROS and Gazebo. Environments developed in OpenAI Gym interact with ROS, which is the connection between gym itself and the Gazebo simulator. The training script uses the get_observation(self) method that allows arbitrary repeated access to the current observation perceived by the agent and publishes this data using the step(self, action) method. This script is the primary control code and users need to edit this code in a manner that fits their requirements and functionalities.

3.1 Installation
We have deployed an automatic installation script known as the helper script to simplify the setup process of the complex stack by a great extent. This bash script upon execution builds the software from source post the sequential installation of all required dependencies, Linux / Python3 packages, platform-dependent / independent tools and available catkin workspace(s). This leads to an easier and faster installation process, which facilitates providing assistance to the non Linux / ROS experienced part of the community.

3.2 Setup
The setup file is a bash script that interacts with the user and upon successful execution automatically integrates the aforementioned plugins with the preliminary ROS package of the robot provided by the software operator.

---

2 https://github.com/ros-controls/ros_controllers.git
3 https://github.com/tu-darmstadt-ros-pkg/hector_gazebo
4 https://github.com/ros-teleop/teleop_twist_keyboard.git
5 http://wiki.ros.org/sw_urdf_exporter
ware operator. This setup file is primarily designed for competition tasks that require a teleoperation task from the point of view of the agent. The keyboard teleoperation package takes input from the keyboard and constantly publishes it to the `geometry_msgs/Twist` message type provided by the differential drive controller. The input to the ROS topic `/cmd_vel` is both the linear and angular component of the velocity via the `geometry_msgs/Twist` message type.

3.3 Graphical User Interface

Owing to the restrictions posed by Gazebo like the inability to view the task environment from the point of view of the agent using on-board cameras present, teleoperation tasks require an additional toolkit for 3D visualization. Luckily, ROS has such a toolkit namely RVIZ. RVIZ is a ROS graphical interface that allows the user to visualize a lot of information using plugins for many kinds of available topics making it suitable for our purpose.

3.4 Command Line Interface (CLI) commands

Most of the times, ending or killing the simulation does not shut out `rosmaster` or `gzserver` and these processes keep rendering output thereby slowing computation and causing the system to crash. `gym-gazebo` installation creates an alias namely `killgazebogym`. This CLI command makes sure to kill all existing background processes before the user starts new tests.

4 Environments

The stack currently provides the following two environments:

- **Leo Rover** is an open-source, all-terrain and waterproof stable mobile robot. The `leo_simulator` ROS package encapsulates the robot’s URDF with a wide variety of environments. Currently, the observation space of the robot is comprised of RGB frames obtained from an onboard camera and data abstracted from an IMU. The action space is comprised of 2 real numbers, the forward velocity and the angular velocity of the rover.

- **lsd_rover** is an open-source package maintained by Mars Rover Manipal. The rover comprises of a custom 5 bar active suspension actuated by motors. The rover is equipped with a LiDAR, an IMU and a stereocamera. The observation space includes the height of the obstacle in front, the pitch angle of the base link i.e. the chassis and an approximate of the obstacle distance from the wheel centre. The action space includes 6 real numbers, 4 of which are inputs to the actuating motors and the other 2 represent the forward and angular velocity of the rover respectively.

5 Experimentation and Results

The aim of this experimentation is to simulate a control code in Gazebo using simple `gym` attributes provided by the developed software stack. The environment chosen is `lsd_force_lidar-v0`. The motive of the control code is to process the observation data provided by the `get_observation(self)` method, feed it into a function approximator and predict the action space data which in turn steps the agent through one timestep in that simulation. The goal of the agent is to successfully climb randomly generated obstacles of varying dimensions by actuating its motors after processing its current observation state. Recently, machine learning has been widely investigated in control problems and applied to vehicle suspension [4], [10], [6], [5],[8], [11]. Owing to the stochastic behaviour of both the agent and the environment, model-free reinforcement learning is chosen to formulate the unsupervised control problem. We have tested several state-of-the-art on and off-policy model-free Reinforcement Learning algorithms like Trust Region Policy Optimization (TRPO) [14], Proximal Policy Optimization [15], Twin Delayed Deep Deterministic Policy Gradient [2], Asynchronous Advantage Actor Critic (A3C) [12] and Deep Deterministic Policy Gradient (DDPG) [9]. As aforementioned, our goal is to develop an algorithm that enables the agent to climb randomly generated obstacles of varying depth and height. The fluctuations about the longitudinal and lateral axis of the base link termed as longitudinal and lateral stability respectively is chosen to be a suitable metric to gauge the performance of the suspension.

---

6(https://github.com/LeoRover)  
7(https://github.com/Mars-Rover-Manipal/Active-Suspension)
The training script is totally independent of the environment. This means that the user can change the algorithm used to learn in the training script, without having to worry about modifying the environment’s structure. Thus, we see that the software developed makes it much easier for a user to simulate a robot environment on ROS using simple gym attributes.

Figure 3: Trained model - PPO & TD3 in action. The figure demonstrates how the trained model enables the rover to climb over a randomly generated obstacle of considerable size.

![Figure 3](image)

Figure 4: Average reward plotted as a function of timesteps.

![Figure 4](image)

6 Future Work

Currently, the developed software stack is being tested to be deployed for the Virtual Mars Rover Challenge. The future work involves us extending the developed toolkit by integrating the gym-gazebo2 [10] stack. gym_gazebo2 provides the same functionalities as gym-gazebo except that it facilitates integration of Gazebo with ROS 2. Furthermore, the setup script currently only generalizes to mobile ground robots. More environments integrated with open source autopilot systems like APM, PX4, ArduPilot, etc will be provided in later versions.

7 Acknowledgements

I would like to thank Mars Rover Manipal (http://marsrovermanipal.com/), for providing me with the necessary resources to successfully conduct this research.
References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. arXiv preprint arXiv:1606.01540, 2016.

[2] Stephen Dankwa and Wenfeng Zheng. Twin-delayed DDPG: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In Proceedings of the 3rd International Conference on Vision, Image and Signal Processing, pages 1–5, 2019.

[3] Scott A DeLoach and Jorge L Valenzuela. An agent-environment interaction model. In International Workshop on Agent-Oriented Software Engineering, pages 1–18. Springer, 2006.

[4] Ikbal Eski and Şahin Yıldırım. Vibration control of vehicle active suspension system using a new robust neural network control system. Simulation Modelling Practice and Theory, 17(5):778–793, 2009.

[5] Ahmad Fares and Ahmad Bani Younes. Online reinforcement learning-based control of an active suspension system using the actor critic approach. Applied Sciences, 10(22):8060, 2020.

[6] Mubeen Khan. Reinforcement learning based suspension damping control system for automotive applications. Master’s thesis, Stanford University, 450 Serra Mall, Stanford, CA 94305, United States, 12 2018. This study explores the application of Reinforcement Learning (RL) methods to a suspension damping control system used to regulate the articulation of the control arm connecting the wheel assembly to the chassis of an ordinary road going automobile.

[7] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), volume 3, pages 2149–2154. IEEE, 2004.

[8] Aleksey Konoiko, Allan Kadhem, Islam Saiful, Navid Ghorbanian, Yahya Zweiri, and M Necip Sahinkaya. Deep learning framework for controlling an active suspension system. Journal of Vibration and Control, 25(17):2316–2329, 2019.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[10] Nestor Gonzalez Lopez, Yue Leire Erro Nuin, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Victor Mayoral Vilches, and Risto Kojcev. Gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. arXiv preprint arXiv:1903.06278, 2019.

[11] Liu Ming, Li Yibin, Rong Xuewen, Zhang Shuaishuai, and Yin Yanfang. Semi-active suspension control based on deep reinforcement learning. IEEE Access, 8:9978–9986, 2020.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[13] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In ICRA workshop on open source software, volume 3, page 5. Kobe, Japan, 2009.

[14] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In International conference on machine learning, pages 1889–1897. PMLR, 2015.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[16] Zhu Si-hong, Lu Bao-zhan, Wang Hui, Zhang Ying, and He Liang. Neural network control method of automotive semi-active air suspension [J]. Journal of Traffic and Transportation Engineering, 4, 2006.

[17] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openAI gym for robotics: a toolkit for reinforcement learning using ros and gazebo. arXiv preprint arXiv:1608.05742, 2016.