Domain Specific Modeling Language for Test World Creation

Abstract:
It is often necessary to use a 3D physics simulator in order to model and test complex robotic systems. Verifying certain behaviors of systems in the real world can be costly, time-consuming, and even dangerous, while simulations are relatively cheap and fast. However, creating simulated environments to test robotic behaviors can take up quite a lot of time and processing power. In order for behaviors to be tested in a variety of scenarios, multiple environments must be created, causing verification time to increase. This paper presents a domain-specific modeling language that can be used to speed up this process. This modeling language can be used in WebGME to generate multiple world and launch files in Gazebo, a 3D dynamics simulator. These world files can then be used to test various behaviors of complex robots such as the CAT Vehicle (Cognitive and Autonomous Test Vehicle) in a variety of simulated environments. This model language can save valuable testing time by quickly creating usable test files for complex physics based models such as the CAT Vehicle.

Introduction: (Jill)

As robotic systems become more complex, it is more important than ever to be able to test them quickly and safely. Many aspects of testing can be completed on a 3D platform, allowing tests to be done quickly and repeatedly. However, even generating this 3D platform takes up valuable testing time and can be difficult to do practically due to limited computing power and time constraints. To remedy this problem, a quick and easy way to generate simulations environments must be created. In this paper, we intend to contribute to this solution through presenting a domain-specific modelling language that can be used to create simulated environments autonomously. This program uses the WebGME meta-modelling environment to generate world files in Gazebo.

Background:

Our approach to creating test worlds focuses on the use of the 3D modeling software Gazebo. Gazebo is a free software commonly used to test simulated robotics due to its physics engine and ability to create complex environments. Gazebo utilizes the Robot Operating System (ROS) to simulate complex robotics that are able to interact with the environment simulated around them (Quigley). Because of this, it is a useful testing tool for researchers to use without necessarily needing a physical platform. Simulated worlds can be created in Gazebo using the virtual environment or by directly coding the environment into a .world file.

The Domain Specific Modeling Language (DSML) for this project was created in WebGME, an online meta-modeling tool (Maroti). WebGME projects consist of a set of models and metamodels in a hierarchical organization (Maroti). The structure of WebGME aids in designing simple, intuitive meta-models, and was ideal for our use. The inheritance based meta models were relatively simple to tie to Gazebo's model and world file structure. WebGME uses JavaScript plug-ins to perform a variety of autonomous tasks across a set of domains (Maroti 52), allowing us to use WebGME's JavaScript code to produce Gazebo compatible C++ code. The implementation and structure of the particular plug-ins used are described in section (Generated World and Launch Files).

DSML - WebGME environment: (Alex)

The modeling environment created is designed for the creation of one or more models of Gazebo worlds, or collections of props with location data and names. The user can choose between two different types of model placement; one takes its location data from the location on the screen that the object is placed, and the other requires manual location data (x, y, z, yaw, pitch, roll). The X and Y coordinates gathered from WebGME by placement location are too large to use, and therefore are divided by ten while being processed by the plugin. The user is required to provide the names for the models that they want to appear in the specified location. In a future implementation, an add-on, list, or plugin can be provided so the user knows whether or not the model exists in their Gazebo's model repository, but this is not provided for this project. Also available for the user to place are preset groups of models (for example, a city block and a thrift store with concrete barriers) that take location data from where they are dragged and dropped. Unlike other WebGME projects, this modeling environment does not use connections between models, but only takes the model's attributes.

Generated World and Launch Files: (Alex)

Upon running a plug-in on the container of the worlds, the location data and model names are used to create world files for the Gazebo simulation program. Also generated are launch files that place the CAT Vehicle into the created world files for the user's convenience. The names of the files and the location data provided by users are processed by the WebGME's plugin and a blank Gazebo world file is used to create a world with models specified by users in their respective locations. Also included in each world file is a statement that includes a custom plugin that, when run by the user, indicates if any of the models they have placed in the world overlap.

The folder also contains a script written in the Python programming language that launches each worldfile with the CAT Vehicle present and following a simulated behavior, which is determined by another Python file provided by us. The CAT Vehicle will, in each world, steer away from obstacles while driving straight. If the CAT Vehicle gets too close to an object, it will stop, and the simulation is considered as failed. The behavior has been tested so that the simulated CAT Vehicle and the real CAT Vehicle run the behavior as similarly as possible. It is up to the user to continue or exit the script after every simulation is run. In future work on this project, the fail or pass state could automatically continue or exit the script, but this is not implemented in this version.

Applications:

Our software was used to create worlds for the CAT Vehicle (Cognitive and Autonomous Test Vehicle), a self-driving car with a well-established testbed in ROS and Gazebo (Bhadani). The testbed allowed us to insert a physics-based gazebo model into our script. Due to the ROS programming, an additional script was added to create a launch file to spawn the CAT vehicle into the world. Using our software, we created a world modeled after a parking lot. After we had created the world, we were able to test a set of behaviors in the world. For our tests we designed a simple Python script that instructed the car to take information from its sensors, drive

towards the nearest object, and then stop a safe distance away. After we had confirmed that these behaviors were safe, we were able to switch to using the actual CAT Vehicle in a real parking lot. Aside from minimal code differences due to the infrastructure of the car, we were able to run the code and watch the car behave as it did in the simulated test world.

We are optimistic that our program can be used to improve testing time for other, diverse projects. We were able to create our parking lot model with minimal measuring and relatively quickly. This could be very useful for researchers that do not have a lot of time to spare for creating test world. It also does not require interacting with Gazebo until after the test world is created, which could help in the case of computers with limited computing power that can normally not handle the load necessary to create worlds by hand in Gazebo. Using our DSML also requires only limited familiarity with the syntax used in the .world Gazebo files

While we are pleased with the way our program behaves, there are several ways our code could be built on. Due to the time constraints, there are additions that could be made that were considered outside the scope of our project. The DSML could be adapted to be more user friendly and aesthetically pleasing. Adding a decorator plugin could help improve the look and feel of the software, as well as making it easier to see overlapping models before the worlds are created. In addition, a plug-in to ensure that behaviors are safe in the generated environments could be useful for testing certain complex robotics. However, within the scope of our project,

Conclusion:

The research question put forth, creating a software to simplify creation and testing of Gazebo worlds, has been met by this project with a WebGME program that accepts models and positions as arguments and outputs worldfiles, launch files, and a script and behavior to test those worldfiles with the CAT Vehicle. The modeling software reduces the need to model in Gazebo and the need to manually create launch files, therefore reducing user time while also providing a template for the user to quickly test any specified vehicle behavior. We hope that the introduction of this software can help future development for the CAT Vehicle and other research projects.

Acknowledgements:

Support for this project was provided by the University of Arizona and the National Science Foundation. The authors express their thanks to the many contributions of students and faculty before them in creating a testbed for the CAT Vehicle. The authors would also like to thank Nancy Emptage for her invaluable support.

*Do we thank the mentors and Dr. Bose here or is it unusual to thank them if they are listed as authors?

References:

* will properly cite all of these when moving over to formatting in latex.

https://arxiv.org/abs/1804.04347v1 - CAT vehicle testbed paper

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler & Andrew Y Ng (2009): ROS: an open-source Robot Operating System. In: ICRA workshop on open source
software, 3.2, Kobe, p. 5

http://ceur-ws.org/Vol-1237/paper5.pdf - Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure - WebGME overview