Virtual Test Course Generation For Self-Driving Cars

Eddie Hew¹,², Muhammad Zuhayr Raghib², Dr. Li Wenchao²

¹Niwot High School, 8989 Niwot Rd, Niwot, CO 80503, ²Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215

Abstract

Although research on self-driving cars is progressing at a rapid pace, an inevitable obstacle in the process is testing. Programs must be installed in real cars to observe actions and bugs, and while fail-safes are present, dangerous and expensive accidents can still occur. The best solution so far are virtual license tests, which puts the AI in a simulated environment to observe its behavior. However, the problem with these tests is the validity of the simulation environment, as it is difficult to emulate all the factors that a self-driving car must analyze in a real life scenario. Thus, the goal of this project was to create an obstacle course for a self-driving car’s AI that would more closely emulate the real world. My solution consisted of three steps: taking a variety of parameters and randomly generating a map that fit those parameters, testing our lab’s AI in the map and recording metrics, and then using those results to create a better map. Using a genetic algorithm, a random population of maps would gradually “evolve” towards a difficult map. This is called adversarial machine learning, and helps us find out crucial flaws.

Program Structure

For Each Map

- Take user input through a file (number of turns, path width, etc.)
- Randomly generate path (% chance of turning) and store in 2D array
- Use recursion to draw walls (repeated models of boxes) around path in the 2D array
- Interpret array into a Gazebo SDF and a path for SLAM

Genetic Algorithm

- Randomly (with starting constraints) generate large population of maps
- Take car’s performance metrics for each population member and generate score
- Generate new population using crossover, mutation, and inheritance
- Run test on new population and refine until desired result.

Genetic Algorithm

An important component of the genetic algorithm is the “parameter space”, or the combination of “genes” that define a child. The planned variables for the parameter space are:
- Number of turns in the course
- Maximum/minimum corridor length
- Path width
- Wall jaggedness
- Obstacle count

Results

- Our current product consists of a program that:
  - Generates a random, one-way path (no intersections)
  - Stays within the memory limit of Gazebo
  - Stays within the space limit of SLAM (2048 x 2048)
  - Takes two parameters, path width and turn amount
  - Outputs a Gazebo world and a .png map for SLAM
  - Has compatibility with both Gazebo 2 and Gazebo 7
  - Outputs to appropriate folders in an organized file system

- Our to-do list includes:
  - Implementing the genetic algorithm to refine difficulty
  - Adding more input parameters to the generator program
  - Packaging the generator and testing software into one program so that tests can run autonomously
  - Adding obstacles and extra models (trees, rocks, etc.)
  - Uploading work to Github

Discussion

Fig. 3: The planned product could look akin to this, but be generated by a computer instead of manually drawn. Color is another planned component of the final product.

- We manually made maps as reference for templates and found it frustrating, so the tool definitely saves hours of manual work.
- Many of the obstacles encountered during the course of research consisted of compatibility and tool-specific problems, so in our master program we tried to make things as seamless as possible.
- The refined version of this program will be able to generate thousands of maps that and thus will be able to test for far more scenarios.
- Coupled with the genetic algorithm, this process should become even more fluid and accurate.
- We also plan to open source this so that the program will continue to improve with time and be a useful tool for years to come.

Conclusions

Our methodology lays the foundation for an evolving map that can teach itself how to be difficult for self-driving cars. In addition, we hope to have the map also teach the cars, and point out places where a self-driving car’s AI could be refined.

Overall, this utility should facilitate faster development cycles and remove the manual process of map-building for those wishing to test their self-driving car AI.

Acknowledgements

Thank you to BU RISE and all involved for organizing this research partnership. Also, thank you to those who worked on resources used in this project.

Tools

Various already developed software was used in the process of our research.

- Eclipse: Java IDE that the generation program is written in. Facilitates packaging of code into applications.
- Gazebo: a 3-D simulation environment that can do physics and lighting calculations, similar to a game engine. Optimized for use with robots.
- SDFFormat: Specialized XML format that describes objects in a Gazebo world. Use of tags makes it easy to manipulate.
- ROS: Robot Operating System. Framework that simplifies code blocks into visual nodes, and handles the car’s behavior.
- SLAM: Simultaneous Localization and Mapping. While ROS deals with moment-to-moment movement, SLAM tells the robot its final destination.

References


Fig. 1: A visual representation of a two variable parameter. Each axis represents one variable. Axis number depends on parameter count.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.

Fig. 2: On left, a bird’s eye view of a generated course in Gazebo. On right, the binary .png that SLAM uses to guide the car through the course.