kuaikai

May 08, 2018

Contents:

CHAPTER 1

Abstract

We propose a system for continuous evaluation of autonomous driving capabilities in high-speed racetracks. In the first implementation, the vehicles used on the racetrack will be small (1/16 or 1/10 of full-size). User-submitted controllers are evaluated remotely on real cars that can complete a racetrack circuit. Lap times are saved to a leaderboard that ranks submissions according to best (smallest) times. Potential users must pass simulation tests and, then, hardware-in-the-loop simulations, before being granted time with the real cars on the racetrack.

1.1 Interface

1.1.1 Introduction

The purpose of this section is to document the interface requirements. Demonstrating examples are available at https://github.com/kuaikai/examples

> **Warning:** Until the first version release, details of the interface can change without notice.

1.1.2 Preliminary concepts

Software submitted by a user is referred to as a **controller**. The entire evaluation of a submitted controller is referred to as a **trial**. Evaluation is performed as a sequence of **jobs**, beginning with simulation. If any job ends with failure, then the trial stops and is marked as failure. Otherwise, measurements from the final job of executing the submitted controller on a freely moving car are saved from completed trials.

1.1.3 Required format and files

The submitted file MUST be a gzip-compressed tar file and MUST contain the following executable shell scripts: build.sh, start.sh, stop.sh, post.sh. Eventually, a bound on the size of submitted files will be fixed, but currently it is 5 MB and might change. If these preconditions are not satisfied, then a trial will not begin.

All shell scripts are run in `bash`. If any script has nonzero exitcode, then the trial ends immediately and is marked as failure. While the user is free to do as she wants, consider the following intended uses of the required shell scripts:

1. `build.sh`: The purpose of build.sh is to build executable files that are used in all evaluation texts. To reduce load on computer resources, the build process should only occur once.
2. `start.sh`: start the controller in each of the evaluation contexts: simulation, hardware-in-the-loop sim, actual car on racetrack.
3. `stop.sh`: stop the controller. The purpose of this script is to allow clean shutdown of the controller when some termination conditions are met, e.g., finally passing the finish line on the racetrack.
4. `post.sh`: run at end of trial, after a successful call to `stop.sh`, e.g., if user wants to upload logs to somewhere off-site

Any other files can be included, at the choice of the participant, but there is a file size bound. Every context in which user-submitted code is running has a time bound. Furthermore, there are bounds on available computational resources, such as memory size. At this time, the bounds are not fixed.

1.1.4 Use of files in evaluation process

Evaluation consists of

1. pure simulation,
2. hardware-in-the-loop simulation, and
3. freely moving car in the racetrack

in that order. The simulation might be on a x86_64 computer, whereas the hardware-in-the-loop platform consists of identical hardware to that of the car model. For the Donkey car, this is a Raspberry Pi with a camera etc.

pure simulation

`build.sh` is run first. After it terminates, `start.sh` is run.

hardware-in-the-loop simulation

`build.sh` is run first. Note that `build.sh` is not run again after this job. It should not be necessary to repeat the build process because the hardware here is the same as in the freely moving car.

After `build.sh` terminates, `start.sh` is run. If terminal conditions are satisfied and the process with PID of `start.sh` is still running, then `stop.sh` is called. If neither terminates within the time bound, then they are killed. If this job is otherwise considered passing, then killing the `start.sh` process is not marked as failure.

freely moving car in the racetrack

`start.sh` is run first. The car must complete N laps in the racetrack. If it goes outside the track, or if a time bound is exceeded, the trial ends with failure. Otherwise, after the circuit is completed, `stop.sh` is called and, eventually if required, the original `start.sh` process is killed. Finally, `post.sh` is run to allow uploading of data for review later.

1.2 Help and contributing

1.2.1 Help

To describe a possible bug or to request a feature, please first check whether it is already reported in the issue tracker. There might also be a pull request that treats it. Otherwise, open an issue.

1.2.2 Contributing code

Thanks for your interest! Before starting development of a substantial new feature, please check whether someone else is already working on it.

There is no official style guide. Try to follow the style manifest in the surrounding code where your change is being made.

1.2.3 Contributing documentation

To facilitate accessible and reproducible results, documentation is critically important in this project.

Documentation is not the place for jokes, political opinions, etc. When there are disputes that do not involve technical issues, consult the guide of Conduct.