Brad Templeton Home

Robocars Main Page

Brad Ideas (My Blog)

Robocar Blog

The Case

Roadmap

Roadblocks

Car Design

Problems

Privacy

Stories

Deliverbots

Whistlecars

End of Transit

Objections

Myths

Geeks

Parking

PRT

Accidents

Notes

Teams

Government

Regulation

Motornet

Urban Planning

Development

Congestion

Definition

Robotic Driving Simulator

A very useful tool on the path to robocars would be a high quality computer simulation of the driving environment -- a sort of "Second Life" for robots. Into this simulation, developers could place their robocar algorithms to see how they do. Fortunately, thanks to the car race video game world, a lot of driving simulation software already exists, and the powerful graphics cards to make it happen are cheap.

The simulation would serve several valuable purposes:

1. It would be a testbed for new robocar software. Software could be developed and tested very cheaply, without the high cost of vehicles, sensors and a test track. In addition, tests could be made in enhanced real world situations, including crazy drivers, higher hazard rates, equipment failures and other things seen only rarely in the real world. All of this could be done without risk to equipment or people.
2. It would be the platform for initial competitions between robocar systems, making it easier for small teams to enter the contests, and making them safer. This would be a springboard for real-world contests among the winners in the virtual world.
3. It is possible to simulate sensors which do not yet exist or which are very expensive. This allows developers to create algorithms that can be used in the real world once the new sensors arrive.
4. A simulator would be valuable for education, allowing students to learn, experiment, and even be graded on their performance.

All complex software projects need a test suite, not just for initial testing, but for regular re-testing. Each time software is modified, if it's easy to test it all over again in an automated fashion, you can be more confident you haven't broken anything with your changes. Test suites try out software in real world situations but also in extreme situations, some possible and some even impossible, to assure it is robust.

Because every team will need such simulation software, and we want everybody to have the best and latest, an open source project for this seems ideal. There are already two open source car racing game platforms, named TORCS and Rigs of Rods which could be used as platforms. Rigs of Rods is less polished, but has a focus on simulating things down the physics which is important, at least for the main car.

It's also possible that these environments can be combined with the existing Gazebo robot simulation environment.

Teams in the Darpa challenges also built their own simulators, and code and ideas from these could also be merged into such a project. But the real reason an open source project might do well is that many people, not just robocar developers, could contribute elements to make the simulator more and more real.

Image from GPL "Rigs of Rods"

More than robocars

The virtual environment could also be used to test other kinds of robots, and even concepts in human driven vehicles, particularly "Intelligent Transportation Systems" tools which aim to provide computerized assistance and data for human driven cars. It could also be used as a testbed for PRT and off-road robots.

Statistics

Glossary

Sidebars:
Charging

Speed Limits

NHTSA Levels

Valley of Danger

Simulator

Google Cars

Sensors

The simulator must simulate a car and its drive-by-wire controls, but also all the sensors that the robot will use.

Cameras

Computer game engines work primarily to build a 3-D world and then present a view into it. That's exactly what is needed to simulate cameras mounted on vehicles. GPU engines can render those camera views very quickly, and the robocar systems would need to attempt to decompose these video streams back into knowledge about their environment using machine vision algorithms. They must take care not to rely on the artifacts of a simulation in doing this.

LIDAR

Fortunately it's easy to generate simulated static LIDAR when you have a complete 3-D representation of the environment, and not too hard to put small errors into the LIDAR readings to match what happens in the real world. This is also true for ultrasound.

Image from TORCS GPL racing simulator

A proper LIDAR simulation will deal with the fact that the sensor and objects are moving while scanning, so different sections of the sweep relate to different points of view. For example the Velodyne LIDAR found on most robocars today has 64 lasers which sweep around 360 degrees every 100ms. At highway speed, that means moving 3 metres each sweep. Because ths is very slow, approximations of these effects are likely.

A proper LIDAR simulation also would return the reflectivity of target points in the near-infrared bands. While most simulators do model the colours of their world in the visible, doing so in the infrared is rare, though estimations can be made from the visible colour. Of particular interest are things like white lines on the road, which are easy to map from visible to infrared.

Radar

Unfortunately radar is reportedly quite hard to simulate in an accurate way. It has a number of strange artifacts that depend on the orientation of surfaces, and multipath reflections.

However, it may be possible to deal with this in a few ways. One is to produce a radar whose errors are not necessarily real, but similar in *effect* to the real errors.

The other may be to look at the best radar interpretation libraries, which try to map radar

results into information about objects and their speeds. The simulator may just return something consistent with that, with the same sorts of errors as that. This means teams can't make use of their own superior radar interpreters unless they can write a simulator for their sort of results.

GPS

Simulated GPS is quite easy, including occasional failures of the GPS, and the typical errors. The source of the errors (multipath, blockages or switching satellites) are not so easy to simulate but the character of them should not be hard.

Accelerometers

Some car racing tools already generate acceleration based on their models of the vehicles and the terrain they are on. Getting realistic bouncing may take some work but seems doable. Inertial guidance is more a matter of introducing the right error into the internal exact position and exact history of vectors the system knows. Fancier systems model the cars as individual components -- at the very least the wheels, suspension and body are modeled as connected components.

Tires & Physics

We would want realistic odometry, but also appropriate models of traction, including skids, drifting and loss of traction on various surfaces in the modeled terrain. Good simulation of potholes, curbs, rough roads and rails embedded in the road are all important.

A good simulation would also model slipstreams, both for drafting, and to measure the effect of being passed by a large vehicle in either direction, as well as gusty windstorms.

Sound

Sound should also be simulated, ranging from people yelling, to sirens, to the sound of vehicle motors and the whoosh of passing. While the use of sound is not common in today's systems, it could become so. Sound also has a PR value, in that it makes watching the simulation more realistic for an audience.

Specialized Sensors

Cars might want to do things like figure out position from wifi MAC addresses and signal strengths, or the signal strengths and IDs of cell towers and broadcasting towers. Of course, it may be simpler just to make an abstract for the position sensing system (since the simulator knows the true coordinates) and just simulate the error of the various systems.

Also worth simulating (or playing back recordings of) will be traffic data feeds, and traffic signal timing feeds (802.11p) as well as other ITS feeds.

Verifier

Not really a sesnsor, the verifier would be a call where the robot could provide to the system an enumeration of all the objects of interest (notably cars, pedestrians, curbs, cyclists, lane makers, signs and traffic signals etc.) that it has identified in a region of space. The system, knowing what is actually present, could compare this enumeration to the "real" list and log any errors for later examination.

In theory, this error report could also be used as a sensor, so that the robot could train itself if it is told immediately about the errors, or even where the error is. There is a risk that such a robot would train itself to identify only virtual objects and not learn correctly about interpreting real sensor data, but this might provide some value if coders are careful.

APIs

The simulator would need to develop an API for use in interfacing with the simulated sensors, as well as one for issuing commands to the virtual drive-by-wire controls of the vehicle. Such an API would probably then be used by makers of the real world equipment.

Controls

Simulating controls provides an interesting challenge. One could start with perfect controls -- you tell the steering wheel to turn, and it turns perfectly to a given angle. You tell the throttle or brake to go to a certain level and it just happens.

Real controls are less perfect. The steering wheel is affected by the wheels and the slope and bumps of the road surface. Real cars must receive information about how the wheel is moving indpendently of the torque applied by a steering motor to the steering column. A good simulation would provide this data.

In general there should be controls for torque on the wheel, electronic throttle and brake control, turn signals, wipers (if you will simulate rain) and lights and any other control a software system might make use of.

Perfect Perception

There is also merit in a system that does not simulate sensors at all, but tells the vehicle what everything is and where it is, the result of a perfect perceptual system within the robot.

The simulator would still model the physics of vehicles, and of course the actions of animated elments in the world. This has several functions:

1. It is much easier to build, and in fact simulations like this already exist and have been used in contests.
2. It is much easier for a team to experiment on only the parts of their code that react to things that have been observed, rather than trying to decode what they are. For individuals, this will be a way to start.
3. Vehicles can be tested with perfect perception in a wide array of circumstances until they make no mistakes. At this milestone, you can begin to conclude the robot will perform nearly perfectly if its perception systems work perfectly, and so work can focus on them.

It's also possible to generate flawed data which still provides a distilled look at the world, but with errors. Simulated perceptual systems can be made to fail at certain times or in certain patterns. The vehicle can then be tested to see how it performs when its perception system is lying to it, or fails to observe objects, or fails intermittently. You can ratchet up the failure level to see how far you can go before the car starts doing unsafe things. Teams can discover what hardware and perceptual system failures can be tolerated and which can't.

Simulated Environmental Elements

Aside from providing a realistic model of streets and terrain (including everything from potholes to slicks from ice, rain, leaves, dirt and other debris) the most important element of the simulation will be other moving things, like cars, people, bicycles, motorcycles, deer, police, scooters and even blowing garbage.

Of these, perhaps the most important and common will be simulators of human driven vehicles. While it is probably not necessary to simulate the physics of these vehicles, you want to have realistic (and exaggerated) human behaviour, including timid drivers, road ragers, drunks, texters, speeders, lane-hoppers, double parkers, slowpokes and everything else that people can code. There has already been much work in this for academic traffic simulations, and this work and new work can be put into the simulator. You want to be able to quickly simulate many thousands of human driven vehicles to make something that looks like rush hour.

Where possible, it would be nice to see simulations derived from real world logs, such as GPS track logs or cameras that watch streets and see patterns. Traffic light data might well be loaded from real traffic light control systems for the real streets that were copied.

Simulations can also be done of other robot vehicles, including ones offered by other developers, and simplified models. These do not need to be nearly so complex as a real robocar driving system, as they get to work with perfect information (the real 3-D model) rather than perceiving it through limited sensors.

Once there is traffic, there can also be traffic reports, which can be used to feed navigation decision systems.

Simulation of pedestrians and cyclists is also important, including jaywalkers, people who walk out from behind parked cars and other obstructions, and even the odd child wandering into the street. People who both obey the law and ignore it must be simulated. Again, this might be fun for independent developers to do.

Terrain is important too, though urban terrain with its curbs and lines and existing maps is fairly easy to decode. (In the first two DARPA challenges, which took place on dirt roads in the desert, it was a big part of the challenge to tell the road surface from open desert.) Deliberately difficult terrain would be a useful contribution. With terrain should also come maps of the terrain (for GPS waypoints) but maps that are deliberately wrong are also of value, as this will happen in the real world.

Wind, as well as the air currents generated by passing vehicles and the results of drafting them, also need some simulation. Here, fully accurate simulation is extremely difficult and so a simplified model will be needed. However, in racing, drafting is a major component, and in driving with bicycles and scooters the wake of a larger vehicle can have a major effect.

Simulation of accidents and dangerous situations is highly useful, as they are not safe to test in the real world. Not only can situations be contributed by programmers around the world, but it should also be possible to recreate a wide variety of real-world accidents which happen to be recorded on video, or detailed well enough in police reports. In Russia and many other countries, so many cars carry dash-cameras that are recording tons of accidents. Human taggers could turn these into models of accidents, and allow simulated robocars to put themselves in these situations and see how they do.

What a simulator can't do

Simulation is highly useful but not a substitute for real-world testing on real roads, or regression testing from recorded real-world data. You can't train machine learning or other trained systems on simulated data or you will just learn the simulated artifacts. Simulated radar is extremely difficult today, effectively impossible for most situations. Simulated LIDAR is possible but to do it perfectly is also very intensive, and LIDAR artifacts are hard to get right.

The real world is full of unanticipated situations, odd lighting, odd angles, strange reflectivities and odd driver behaviour that you won't find in sim. Simulation is a good way to get the basics down, and to test situations too dangerous for the real world, but it must be done in parallel with real testing. Testing on recorded data tests better the results in real situations, but the vehicle can not test its reactions on recorded data, only that it reacted as it did when the recording was done.

Networked Simulation

For all purposes, but especially for testing, it would be useful to build a simulation that can network. In this fashion, each robocar would use its own computing power to render the world and produce things like virtual cameras and LIDAR, but the operation of moving objects in the world would be broadcast from a central computer. The actions of robocars running in the simulator would be sent up to the master computer to distribute to everybody else, so driving would take place in a big shared world Low latency would be important, at least for contests, which would probably be held in one facility on a gigabit LAN. Many cooperative runs could probably take place on the regular internet or internet2.

A networked approach also allows the system to scale, so that operations are distributed, and it becomes possible to generate a complete simulated city, with a million simulated cars and pedestrians wandering around it. Anybody with a GPU able to receive the broadcasts could then see it in real time on their screen.

Of course, for those doing testing in their own labs, they could run it all on one computer, or on a small local network. If a car needs so many cameras that one GPU can't handle things, this architecture allows adding extra computers to generate more virtual cameras or other virtual sensors.

Getting one open source robot

The project -- and the robocar world -- would be greatly facilitiated by having at least one decent quality open source robot. Having such a robot would be greatly useful for the testing and refinement of the simulator itself, and provides a means of having other advanced robot vehicles on the road for those testing their own code. The open source robot would also be a likely starting point for experimenters trying to build their own robots.

It is hoped that some team, perhaps one of the DARPA challenge teams, will be willing to donate their code, even in an older version, to such a project. Some of the Junior code (which placed a close 2nd in the urban challenge) is available open source.

Avoiding artifacts and bumps

Simulated sensors, particularly simulated cameras, are still artificial. Humans can tell the difference in an instant. Unfortunately, machine vision systems will also be able to tell the difference, and some may pay attention to the artifacts of a VR image to do a better job of finding edges or interpreting information from the scene. This is something we don't want in testing and can't have in a contest.

While nobody would deliberately exploit artifacts in testing, some machine vision systems are based on learning by processing lots of video input. These learning systems might be fooled by artificial images and end up not working on real world camera images. Developers would need to take care to avoid this.

To help, contributors could drive real world routes which are also available in the simulation, recording them with real world sensors. Then they could take the tracks generated from GPS and other positioning tools from the real world run, and re-generate them in the simulator. Developers could then test to see if their algorithms work as well on the real world data as they do on the simulated ones.

In contests, people might actually deliberately look for artifacts to get an edge. To stop this, they may need to reveal their source code so that it can be examined for cheating. In addition, the contest team could develop new parallel streams (from real world and simulation) that have never been seen by contestants, and assure the contestants can also handle the real world data.

Real world cameras and other sensors will also show a bumpy ride. In past contest, some competitors tried to do physical stabilization of their cameras, others used software image stabilization. The latter of course will test out fine in a simulated world, given simulated accelerometer readings or similar data.

It makes sense for the simulator to also offer "cleaned up" or perfect data for testing. This allows new developers to get going sooner, either with the assumption they will work on handing the jitters of real data later, or they will depend on using outside code (or machinery) to clean up data when the time comes.

For contests, it might make sense to give developers a "budget" and to then put a price on all sensors and features they ask for. If they want pre-stabilized cameras that will cost them more money. Having extra cameras, radars or LIDARs will cost more. Having better suspension or a better engine might cost. In such contests, the goal would be to get the best performance for a fixed amount of spending.

Simulators can offer developers sensors which might be quite expensive to get in today's world. For example, a fully-stabilized instant target-and-zoom camera is easy to do in simulation. This might be used, for example, to zoom in on the faces of simulated pedestrians after they have been generally identified, to either confirm their identity, or even to examine their facial expressions and where their eyes are looking. (This is already a popular problem in machine vision.)

Finally, simulated algorithms can be offered perfect "distilled" sensors, such as ones that report, "There is an SUV at these coordinates moving on this vector." Such "sensors" would be used by teams working on what to do with the data once it has been turned into a map of the world by theoretical future systems.

Proprietary and Open

An open system, with everybody contributing, should work well, particularly because many of the elements are fun, or are the subject of academic study -- academics don't mind so much contributing for free to shared projects.

Some developers may want to keep things proprietary. Under GPL licence rules, they can do this but if they ever want to copy their new work to others they must publish the source code. (This does not mean they could not sell the code they tested using the simulator.)

There is no denying however, that once robocar development moves beyond contests and becomes commercial, there will be strong commercial demand for high quality simulators. This may create conflict between the open and proprietary worlds. This may even be true today, as car manufacturers are already working on robocar-related safety systems, such as Volvo's S60 pedestrian detector, and good simulators would be highly valuable in such projects.

Related Projects

In addition to TORCS and Rigs of Rods, there are other simulators for robots, some with adoptions for roads. These include:

- USARSim The Unified System for Automation and Robot Simulation -- this hosts the RoboCup virtual rescue contest.
- A team in New Jersey has a project on github based on the Unity3D game engine.
- People have hooked Wilow Garage's ROS (Robot Operating System) into simulation tools like Gazebo.
- Another robot simulator based on Gazebo is Stage -- it even has precompiled binary packages.
- MORSE is another academic open robot simulator.
- breve is a 3-d world simulator which can be used for robots. I don't know if it has been used for driving.
- OpenSimulator is a shared virtual world platform, best described as an open system with similarities to 2nd Life. It is written in C#/.net/mono which might present issues in GPU use in simulation of LIDAR and other sensors. It otherwise seems ideal.
- Every year, a virtual car race and virtual demolition derby are held in a modified version of TORCS. This is done at the logical level, the software is told the exact ground truth about where other cars are and where the road is.
- An effort is underway called PyTorcs to redo Torcs in Python.
- The Dutch lab TNO has a full sensor level driving simulation used by car vendors to test their driver-assist systems. It is called Prescan but is an expensive, proprietary commercial system based on Matlab and Simulink.
- Another commercial simulator is carSIM however it is not focused on robot sensors.
- The new online University Udacity is offering a course on creating self-driving car software. Some of the students are interested in a simulator. It would be great if those students could test and compete with their virtual cars in a simulator.

Thanks to Keith Curtis for some of the initial ideas in this area, particularly the general concept of an open source VR world for use in contests based on video games.

Comments can be left at this blog post.