Game Engine-based Agent Simulator (GEAS)

UMCS Senior Project '13

Alexander B. Strout
Roy M. Turner

Abstract

All software requires a reliable testing environment for development, and AI control software on an autonomous vehicle is no different. One possible environment for this might be the production environment - the end-user's system, the live web servers; or in this case, a physical autonomous vehicle. However, there are several reasons why this is often not ideal - errors in the software that crash the end-user's system (or the autonomous vehicle) are costly, and continuously pushing updates to such an environment may be slow and tedious. Within the realm of autonomous vehicle development specifically, high vehicle costs and limited availability often suggest the design of computer simulations as testing environments that are simultaneously more flexible and feasible than physical vehicles.

However, existing computer simulation environments for autonomous vehicles carry their own share of issues. Typically developed from the ground up for the particular vehicle they are intended to simulate, they are difficult to reuse without significant retrofit. Additionally, systems common to many of these simulations - such as graphics rendering, physics simulation and networking support - are often also developed for these platforms, demanding significant development time just to prepare the basic 3D world the vehicle will be simulated in.

This project aims to construct an autonomous vehicle simulation framework using advanced 3D environments: game engines. Game engines provide the necessary capabilities to construct detailed 3D interactive worlds with off-the-shelf rendering and physics engines available, eliminating the need to "re-invent the wheel" and allowing designers to move straight into making games - or, in this case, autonomous vehicle simulators. The main focus of this project's framework is to provide an interface to allow existing autonomous vehicle control software to control simulated vehicles within the game world.

Introduction

Design of AI systems for autonomous vehicles is a complex process that is tightly bound to - and can not reasonably be done independently of - the vehicle being designed for. The best solution for this is to of course have one or more real vehicles present and ready to use where the AI system research and development is being conducted. However, there are many reasons why this is infeasible: first, AI researchers may not necessarily be prepared to handle the maintenance and operation of the vehicle, particularly on large-scale projects. Second, purchase and maintenance of a vehicle is typically costly. Third, errors in the AI control software could potentially damage the vehicle, incurring further costs. Finally, if applicable, the vehicle may be better purposed for deployment using existing AI control software.

As such, the most reasonable development solution is typically a computer simulation. Such a simulation must attempt to accurately emulate real-world conditions and equipment -

such as radar, lidar, and sonar sensors - to allow development of AI vehicle systems in a virtual space without incurring the costs associated with operating and maintaining a physical vehicle. However, as autonomous vehicle development is often highly specific to the vehicle itself, the availability of off-the-shelf simulations to suit the needs of a particular project is slim. Instead, these simulations are typically built from the ground up, with the emulated vehicle systems implemented by the AI researchers themselves. In the case of autonomous underwater vehicles (AUVs), examples might include the CADCON[7] and SMART[8] systems. Unfortunately, this also typically involves "overhead" design of rendering, physics, networking or language systems, which can become significant projects in and of themselves. Additionally, these systems become quickly antiquated, demanding continued time and attention from researchers that could be better spent toward further AI development.

Fortunately, there are alternatives - numerous off-the-shelf platforms with capable rendering, physics and networking systems which are becoming increasingly - and in many cases freely - available: game engines. As the name might imply, these game engines are typically built for the design and development of modern 3D video games. Examples might include Unity[1], Unreal[2], Torque3D[3], Polycode[4] and CryEngine3[5], which range anywhere from well-maintained rendering and physics libraries to full toolkits featuring GUI-driven game world and object creation suites.

The Game-engine Based Agent Simulator (GEAS) aims to instead leverage these platforms for simulation development using the freely available Unreal Development Kit (UDK). Although specifically aimed at AUVs, the main goal of GEAS is to provide a framework for autonomous vehicle research and development through a flexible interface to allow existing intelligent control software to communicate with in-game objects, allowing such objects (in this case, our simulated vehicles) to be "driven" by the external control software.

GEAS: Related Work

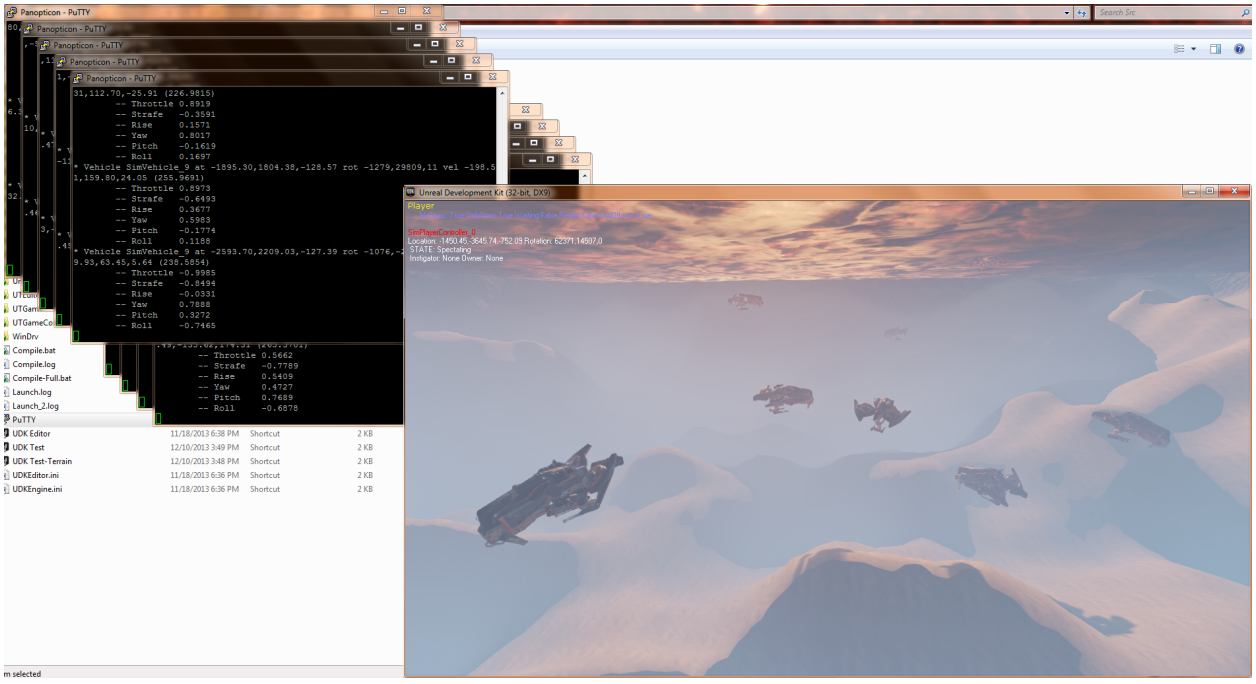
A good example of prior work is Brutzman's DIS-Java-VRML[6] simulation system, written entirely in Java using the DIS protocol. The goals between this project and VRML are quite similar; to provide a functional simulation environment for real-world AI controllers. While it was a very advanced simulation (for its time), it was also entirely written from the ground up, and its age clearly shows (at least visually). By decoupling the DIS implementation from the "game" properties of the simulation (physics, rendering, etc.), individual parts of the simulation can be much more easily swapped out without having to re-write unnecessary parts (for example, another translator could be written for this project that handles Open Sound Control protocol, and the in-game simulation wouldn't care as long as it received the expected traffic over the socket connections).

GEAS: Control

In common modern video games, interactive objects - or *actors* - are typically controlled either by one or more users (the game players), or by scripts run within the game world itself. The code objects that handle these inputs are commonly called *controllers*. However, for the purposes of AUV simulation, we are primarily concerned with allowing existing AUV software to drive the simulated in-game AUV - thus, the in-game AUV's controller and external AUV control software will need a way to interface with each other.

To this end, we have the first and most important aspect of the project - control. Instead of using in-game input to drive the simulated SUV, commands are sent over a socket connection and resolved to in-game controls - such as throttle and steering axes. In return, emulated sensory data from the in-game AUV actor is relayed back to the external AUV control software.

Multiple in-game AUV and external controller bindings are supported. Currently, when a new socket connection is opened, an appropriate AUV actor and controller are spawned and linked to that open connection. Input commands or queries for sensor data made over that connection will communicate with the in-game AUV independently of any others active.

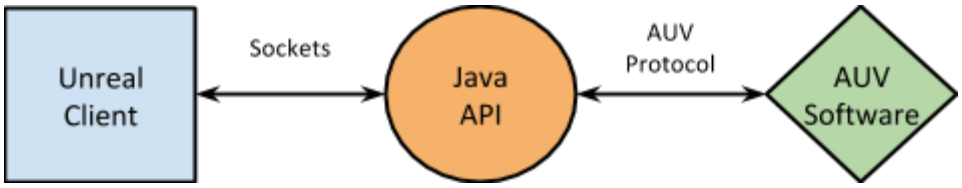


Multiple simulated AUVs with corresponding control connections.

Unreal supports TCP sockets via the TcpLink class,[1] implemented within a general purpose socket communication interface. Utilizing the socket interface is a template Unreal

[1] Unreal TCPLink info: http://udn.epicgames.com/Three/TcpLink.html

controller to execute instructions on the in-game simulated AUV based on the translator / external AUV control software's input. AUV control software can either directly open a socket connection to the Unreal simulation or, alternatively, utilize a simple Java API available.



When first designing the project, one foreseeable issue was that it was not currently known whether TCP socket connections would in reality prove an adequately reliable and responsive interface between the AUV software and the simulated AUV within the Unreal game world. This was the preferred method of communication, as while Unreal is capable of binding DLLs for running native code within classes,[2] using a networked protocol was more versatile and allowed the control software and simulation to be running on different machines. However, socket control has so far proven reliable under a variety of setups.

GEAS: Unreal AUV Simulation .

Of course, there's no sense building a simulation framework without a demo simulation. The next component of GEAS is implementation of AUV simulation within Unreal. This includes modelling an AUV actor that resembles its real-world AUV counterpart's physical traits and sensor suite as accurately as possible.

Simulated AUV using provided Unreal sample content

In modeling the AUV sensor suite, we currently model GPS and Depth sensing capabilities, with emulation for Radar, Lidar, and Sonar sensors currently in the works. The GPS

[2] Unreal DLLBind info: http://udn.epicgames.com/Three/DLLBind.html

sensor simply reads the AUV's world coordinates, which the Depth sensor uses to calculate the distance from the water level. Lidar information is obtained via "traces" from the actor - essentially firing a bullet and determining distance based on where the bullet hits. Radar and Sonar can be primitively emulated with time-delayed queries of all object positions, reporting those within the sensor range.

Currently, multiple Unreal clients can connect to and view a single simulated world in real-time, with the events occurring in the simulated world synchronized across all clients. By design, no direct interaction is possible with the world - clients are merely "spectators," only present to view the game world. Clients may freely fly around the game world to gain a better view or, alternatively, cycle through views of all active AUVs within the game world.

The Unreal simulation also includes primitive modeling of an underwater environment, including an underwater terrain mesh that may be modified using Unreal's terrain tools. There is also an established sea level with visible water surface that the AUV is not allowed to bypass. Underwater movement is currently simulated with a free-floating AUV actor, which can move as desired before colliding with either the seabed terrain or water surface.

Underwater world snapshots

While underwater environments in video games are typically simulated with lower movement speeds and decreased gravity, we're interested in simulating as many factors as possible, and future work may include simulation of currents, variable temperature zones, and water pressure. Currents may be emulated with pseudo-random physics impulses applied to the AUV actor over time. Temperature variance, and its effect on simulated hardware, may be modeled as a pseudo-random function of world coordinates.

GEAS: Future Work .

Future work for GEAS may include:

- Adjustable parameters for the simulation, such as environment properties like weather (as well as associated effects on underwater currents)

- Ambient in-world agents (such as fish and other oceanic wildlife)
- Additional sample environments and agents

Future work for GEAS will **not** include:

- Custom graphical assets for demo simulations
  (except for the world itself, which was created within Unreal)
- Direct interaction with agents
- External controller development
- Out-of-engine world definition
  (environments and actors must be defined within Unreal)

GEAS: Evaluation

GEAS should be evaluated on the following merits:

- *Performance*
  - Can the Unreal simulation perform adequately across a range of hardware?
  - Can the in-game simulated AUV be adequately controlled by external input?
- *Portability*
  - Is the simulation system easily installed and run on a wide range of hardware?
- *Generality*
  - How easily may the framework be used for other types of autonomous vehicles?
- *Simplicity*
  - Is the simulation easy and sensible for researchers to utilize?

References

[1] Unity3D web site. http://unity3d.com/. Accessed 10/03/2013.

[2] Unreal Engine web site. http://www.unrealengine.com/. Accessed 10/03/2013.

[3] Torque3D web site. http://www.garagegames.com/products/torque-3d/. Accessed 10/03/2013.

[4] Polycode web site. http://polycode.org/. Accessed 10/03/2013.

[5] CryEngine3 web site. http://www.crytek.com/cryengine/. Accessed 10/03/2013.

[6] DIS-Java-VRML web site. http://faculty.nps.edu/brutzman/vrtp/dis-java-vrml/. Accessed 10/03/2013.

[7] Steven G. Chappell, Rick J. Komerska, Liang Peng, Yingchun Lu. *Cooperative AUV Development Concept (CADCON)*. In *Proceedings of the 11th International Symposium on Unmanned Untethered Submersible Technology (UUST '99)*, Durham, NH, August 1999.

[8] R. M. Turner, J. S. Fox, E. H. Turner, D. R. Blidberg. *Multiple Autonomous Vehicle Imaging System (MAVIS)*. In *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology (UUST '91)*, pages 526-536, Durham, NH, 1991.