Hardware-in-the-Loop

Related terms:

Embedded Software, Software Verification, Software Testing, Testbed, Embedded Systems, Software Engineering

View all Topics

Model Development and Verification

George Ellis, in Control System Design Guide (Fourth Edition), 2012

13.2.6 Hardware in the Loop

Hardware in the loop (HIL) or Hardware-in-the-loop-simulation-testing (HILST) is, in a sense, the opposite of RCP: the control laws are deployed to final hardware while the power converter, plant, and feedback sensors are simulated. As with RCP, the simulation must be executed in real time because part of the system (here, the controller) is physical hardware. Also, there must be hardware to interface to the control law output and feedback sensor signals. This configuration is shown in Figure 13.12.

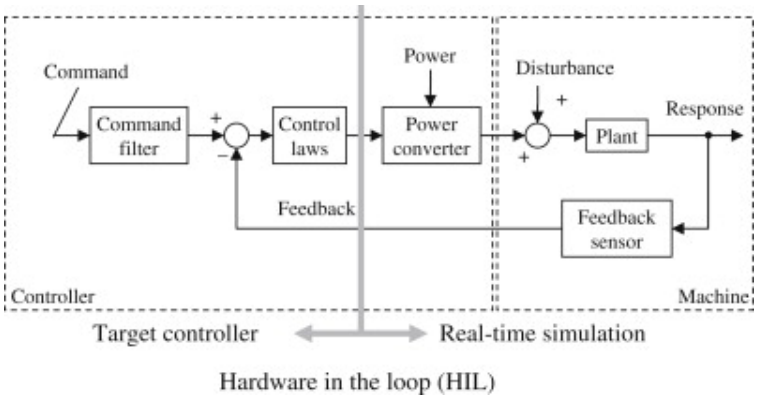

Figure 13.12. Hardware in the loop (HIL).

HIL has been used for years in industries such as aerospace or automotive where plants are especially complex. HIL can augment testing of physical plants to provide many advantages such as:

- Reduced cost to testThe high cost to execute tests on complex machinery like aircraft sub-systems can justify substantial investment in HIL. Field testing can be reduced by using HIL as a complement. On their Web site, National Instruments presents one example where Process Automation reduced field test time by half using HIL, saving $150 K in costs associated with testing an aircraft arrestor control system (www.ni.com/HIL, click "What is HIL Testing?").
- Reduce risks associated with failureWith complex plants, control system malfunction can lead to catastrophic failure, destroying equipment or presenting safety hazards. HIL can be used to validate controllers before running physical equipment. This can be used at the start of the project to validate a new controller; it can also be used throughout development to reduce the chances of software changes to the controller introducing new failure modes (often referred to as *regression testing*). Both uses reduce the likelihood of encountering unexpected failure modes on physical hardware.
- Concurrent development of components of the control systemIn many control system development projects, the controller may be available long before the power converter, plant, and feedback sensors are available. Using HIL, testing on the production controller can begin before the other components are ready. This can reduce overall development time while increasing confidence in the controller.
- Testing many plant variationsPlants often vary, and testing samples of each variation can be expensive. For example, windmill turbines are available in many power sizes. If each test cycle has to be carried out on every variation, so many versions of the plant may have to be constructed for testing that costs are prohibitive. Using HIL, testing from a subset of the available variations can be leveraged. For example, field execution of some tests could be carried out on a few variations (say, the largest, smallest, and a few other models). Then HIL testing could be used for some functions of the intermediate models.
- Testing fault modesHIL allows more robust testing of fault modes. Often thorough testing of fault modes is impractical when relying wholly on physical systems. With HIL, faults can be induced through software and can be synchronized with a wide range of conditions. For example, a brake fault could be simulated when the HIL model of the vehicle is operating in a great number of combinations of speed, vehicle weight, and various ambient conditions. For faults that arise from mechanical failure, inducing faults over such a wide range of conditions on a physical brake might be impractical.

An example of HIL used to speed development and improve safety can be taken from the EcoCAR Challenge (http://www.ecocarchallenge.org/). In the EcoCAR challenge, teams from universities compete to produce the most fuel efficient automobile prototypes. A recent survey showed numerous teams relying on HIL. For example,

Virginia Tech and Ohio State University, two of the best performing teams in the EcoCAR challenge, heavily relied on HIL for success in the competition. Both teams ran vehicle and engine controllers connected to HIL simulation equipment from different vendors — Virginia Tech using National Instruments products and Ohio State University using dSPACE products. Both reported similar results: schedules are compressed because testing can begin before a power train is installed in the vehicle.

> Read full chapter

User Friendly Environment for the R&D of Controllers in Heavy Machinery

T. Virvalo, ... M. Kivikoski, in Human Friendly Mechatronics, 2001

5 CONCLUSIONS AND FUTURE RESEARCH

The results show that HIL is a good approach for testing distributed control systems. It is easy to generate different loading conditions in the simulation model and therefore the controller hardware can be tested in all necessary situations, some of which might be even dangerous in the real system. It can be expected that the HIL approach remarkably accelerates the testing of control systems.

The research will continue. The simulation model should be improved, and each controller also requires a lot of R&D. The reliability and performance of the distributed control system, including e.g. the analysing of delays, timing and drifts of the controllers clock in the CAN bus system require a lot of R&D, too.

> Read full chapter

Testing of Intelligent Vehicles Using Virtual Environments and Staged Scenarios

Arda Kurt, ... Ümit Özgüner, in Advances in Intelligent Vehicles, 2014

2.1 Introduction

Sufficient testing is an essential part of engineering research and development. For our main focus, Intelligent Transportation Systems (ITS), the testing phase often includes preliminary testing in numerous simulation environments as well as physical tests involving one or more vehicles.

Incorporating virtual equipment in the testing and evaluation of controlled engineering systems is now commonplace, and the availability of powerful computation and data acquisition and control equipment has made hardware-in-the-loop something of a standard practice. These approaches are also being employed in the design and testing of intelligent and automated vehicle technologies. There are a number of factors that justify this approach, including:

- The cost or availability of equipment
- The time and effort required to manage and staff field test activities
- Risk and safety considerations, especially with unproven technologies.

To the best of our knowledge, the earliest uses of virtual hardware in the area of intelligent transportation systems appears in the work of Robert Fenton, beginning in the late 1960s. Fenton was one of the earliest active researchers in the area of automated vehicles and driver–vehicle interactions.

In Ref. [1], he describes a longitudinal control strategy for car following, what we would call platooning in an automated highway system, the implementation of this controller on a drive-by-wire experimental vehicle, and experiments conducted at highway speeds. Because there was no practical method or sensor available to measure the separation distance (headway) to the lead vehicle and the velocity or relative velocity of the lead vehicle, he created a simulated "phantom car", whose behavior could be varied according to the desired test, to produce the required sensor readings for his control experiments.

In Ref. [2], he describes an early driver-assist technology for vehicle following. He designed a haptic interface to provide information about lead car headway and relative velocity to the human driver and experimentally demonstrates the resulting improvement in car-following performance versus an unaided driver, especially in short-headway situations. In addition to the practical sensor issues faced in Ref. [1], these experiments involved human subjects not affiliated with the research program, so the use of a simulated lead vehicle during initial trials and while training test subjects was an important safety factor and training aid.

One major limitation of tests involving multi-vehicle scenarios, as we observed throughout our more recent experience in this particular field [3–5], is that outdoor testing can have a high cost in terms of logistics and scheduling, as mentioned above. Since adequately equipped testing areas are not always readily accessible,

scheduling for transportation of the multiple parties involved and for favorable weather conditions can be a problem in and of itself.

In order to mitigate some of the logistics, cost, and safety problems listed above, this chapter describes a low-cost, flexible supplement to outdoor testing for intelligent transportation research and applications. Starting with fully virtual computer simulations and gradually increasing the real-life components of the tested scenarios, complex cyber-physical systems can be studied without many of the associated costs of actual outdoor testing.

For realistic urban traffic, where autonomous vehicles interact with human-driven vehicles, the aforementioned complexities of outdoor tests can be even more daunting; therefore, an indoor testbed that emulates the focused aspects of an outdoor environment is often effective for tests involving multiple vehicles, higher-level decision-making, and situation assessment. The generally lower speeds of urban traffic scenarios, as opposed to automated highway systems, makes indoor testing a particularly attractive option for consistent, repeatable tests.

The indoor testbed at The Ohio State University Control and Intelligent Transportation Research Laboratory [6], named SimVille and seen in Figure 2.1, has served as the intermediate, semi-virtual test environment between computer simulations and fully developed outdoor tests for a number of projects over the years.

Figure 2.1. SimVille Indoor Testbed at OSU Control and Intelligent Transportation Research Lab, Through Various Incarnations and with Different Mobile Robots.

In the following subsections, the concepts behind the semi-virtualized iterations of the testing procedure, including simulations, small-scale testbeds, and full-scale outdoor testing, will be described and examples will be used as illustrations of these.

> Read full chapter

Functional and Nonfunctional Design Verification for Embedded Software Systems

Arnab Ray, ... Chris Martin, in Advances in Computers, 2011

2.2 Functional Verification

A *functional requirement* is a statement of what the system must or must not do, usually expressed in the form: if a given condition holds, then the system should respond appropriately. *Functional verification* consists of checking whether the software satisfies the functional requirements.

In projects that follow MBD, code is automatically generated from models and then flashed onto the hardware. Tests are then run on the hardware–software combination (known as hardware-in-the-loop testing) and if the tests pass, the system is considered to have been successfully verified.

Hardware-in-the-loop testing is an expensive and time-consuming process not only because physical test beds and environments are difficult to maintain but also because errors that are discovered postcode generation are tougher to rectify (in terms of time and cost) than if they had been caught in the modeling phase itself. The principal criticism of existing practices in functional verification of models is that the executable property of modeling notations is not properly leveraged. Since models can be run like code, a significant part of functional verification can potentially be pushed upstream in the development cycle.

A variety of techniques have been developed to provide automated and semiautomated methods for model verification that take advantage of the execution semantics of modeling notations. These include (1) guided simulation [31] where scenarios deduced from the use cases are "run" on the model and the outputs produced studied to see if they are what was expected (2) model-checking, an automated technique which given a model and a property expressed in a logical language determines if the model satisfies the property and if it does not supplies the user with a counter example (3) deductive reasoning wherein theorems are sought to be proven on software models.

> Read full chapter

Introduction

Daniel Aarno, Jakob Engblom, in Software and System Development using Virtual Platforms, 2015

Integration Testing

The classic MDD flow does not really touch on the issue of system integration. The focus is on generating correct code for the core functionality of the system. That code will need an operating system and a hardware platform to run in the real world, and integration with that platform often happens quite late in the system design cycle. Indeed, even HIL testing is often performed using development hardware rather than the final system.

Simics can be used to move integration earlier and allow earlier testing of the integration. As shown in Figure 1.6, with a Simics model of the target system, the OS port to the target hardware and the integration of the operating system and applications can be tested without hardware.

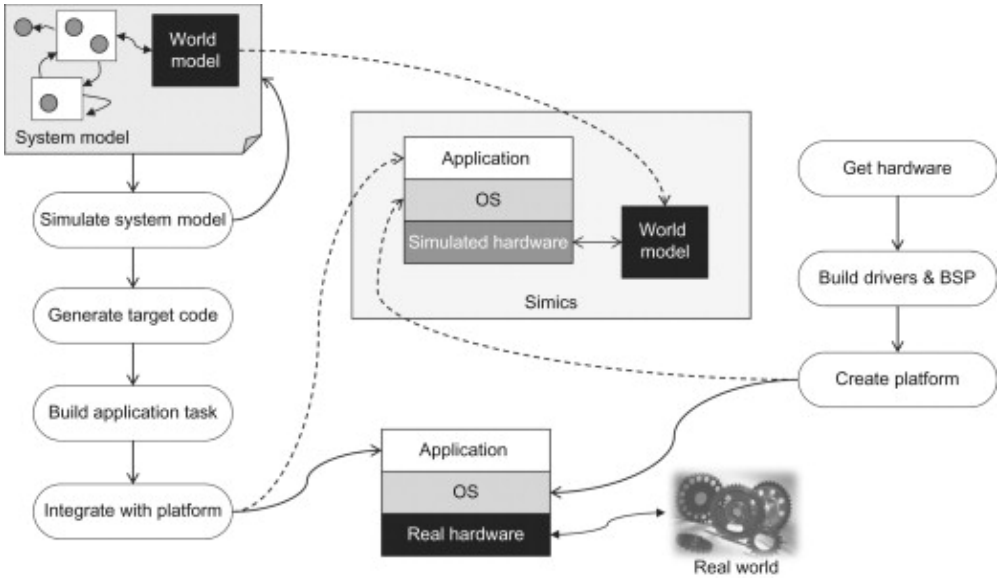

Figure 1.6. Model integration testing with Simics.

Something that Simics makes possible is to test that the application works with the actual input and output as provided by the operating system and target hardware platform, while still using a model of the world. Thus, it is possible to create a fully virtual integration environment, where hardware, the operating system, the applications containing control algorithms, and the world can all be run together in simulation.

> Read full chapter

Effect of Human Factors on Driver Behavior

Jianqiang Wang, ... Xiao-Yun Lu, in Advances in Intelligent Vehicles, 2014

Abstract

Driver behavior is one of the most important aspects in the design, development, and application of Advanced Driving Assistance Systems (ADAS) and Intelligent Transportation Systems (ITS), which can be affected by many factors. To study the effect of human factors in driving, a simulator with Hardware-in-the-Loop (HIL) and Driver-in-the-Loop (DIL), and an instrumented vehicle were developed to quantitatively measure the driver actions, vehicle status, and the relationship with the vehicle. A comparative analysis of the differences was carried out in a driving simulator and also in the real-world environment. After 33 various driver experiments were conducted, a set of special data-processing programs were also developed to review, extract, and analyze the captured data. Some representative parameters such as time to collision (TTC) and time headway (THW) are adopted to analyze and compare the effect of human factors, which include age, gender, driving experience, workload, education level, and nationality. Finally, to ensure the consistency of classifications based on subjective evaluation and objective experimental data, tests including the completion of a driver behavior questionnaire (DBQ) and real-world driving were conducted with 52 participants and compared. Factor analysis theory was applied to the data processing and characteristic extractions. The results of the study indicated that there were significant differences between the subjective evaluation and objective data, i.e. the subjective evaluation was not reliable enough for algorithm design, although it can be used as a reference.

> Read full chapter

Using Simulation for Research

Thomas W. Edgar, David O. Manz, in Research Methods for Cyber Security, 2017

Instantiating a Model

As we have previously mentioned, a model is an abstract or conceptual representation of a system. Models in and of themselves are not executable. Instead, a tool is needed to turn a model into an executable process that generates data. Simulation is the process by which a model is turned from descriptive into infor-

mative. Simulations require a programming framework through which a model can be specified and then an executable platform that can run the model under various specified conditions. A simulation runs through scenarios of inputs with a model to generate data about the state of the model and the output that would be produced. For example, one of the common types of simulators we will discuss is a network simulator. Network simulators take a model of a network and allow the simulation of behavior as communication passes through it. Network simulators enable researchers to study new protocols such as how a new wireless protocol will function under different configurations of a network.

Types of Simulation

There are several ways to define a model and different simulation platforms. **Agent-based simulation** uses models of agents that operate independently. These types of simulations are good for discovering emergent behavior when there is distributed control of a system. **Process-based simulations** use a mathematical definition of a system to generate behavior from a process. Process-based simulations are good for simulating physical processes that can be defined with continuous mathematics. **Trace-based simulations** execute small-scale applications to generate logs of real behavior, and then the simulation framework interpolates these logs to a much bigger scale. Trace-based simulations are good for understanding scalability performance of systems. Finally, Monte Carlo simulations execute a number of tests to determine the likelihood of outcomes. Monte Carlo simulations are good for decision support or when there is a large amount of uncertainty on the input parameters.

Each of these approaches have trade-offs. Some provide easier modeling. For instance, modeling independent agents is easier than modeling a full system with interdependencies. Decisions you make on models affects the performance and scalability. A matrix-based model like the power grid is not easy to decompose into sub problems, which makes it difficult to scale.

While these are general ways of simulating systems, there are specific types of simulation that are useful for cyber security research. The following list provides a good overview of different types of simulation that might be useful in helping answer your research questions.

Network simulators

- *Container-based simulators*: Container-based simulators leverage lightweight virtualization to enable the generation of a network of communicating nodes on one or a small number of physical computers. Container-based simulators are good at looking at new network protocol behaviors. However, as all

containers will leverage the exact same communication stack, they are not a ■ good simulator for tests of heterogeneity of unique response behavior. These simulators have been created for software-defined networking,[11] wireless applications,[12] and general networking.[13]

Discrete event simulators: Discrete event simulators generate sequences of discrete events. As cyber space is a discrete space, discrete event simulators are well suited for simulating aspects of it. In particular, discrete event simulators have been the main method of simulating networks and protocols. There are multiple discrete event network simulators; commercial[14,15] and open source.[16,17] Each have unique characteristics such as configuration and analysis graphical user interface (GUIs), network emulation features for hardware-in-the-loop, and various supported protocol and physical models.

Dig Deeper: **Hardware-in-the-loop**

Hardware-in-the-loop is a special kind of simulation that fuzzes the lines with experimentation. Hardware-in-the-loop simulation provides emulation capabilities that enable the integration of real equipment into the simulation. Generally a larger system is simulated with a few real devices to enable the high fidelity response of real equipment while enabling the scale of the simulation. One of the main reasons behind the redesign of ns-2 into ns-3 was to enable hardware-in-the-loop testing to enable quicker progression of evaluating real implementations.

Traffic Generators

Where network simulators attempt to model and capture the realistic behavior and output of communication networks, traffic generators are generally only concerned with modeling and simulating the communication packets and payloads that would be produced by devices on a network. Or stated another way, traffic generators simulate many devices on a network and the communication they would produce, but they are not concerned with measuring that traffic across the network. Traffic generators are often used for applied experimentation to investigate the performance of network infrastructure, sensors, and security controls. There are general traffic simulators that generate various protocols and behaviors.[18,19] Specifically for security applications there are various traffic generation tools that generate attack behaviors.[20,21,22] There are also higher fidelity traffic generators that, instead of simulating traffic, emulate user behavior to generate real traffic from real applications.[23,24] Emulated users are useful both for applied experiments and fundamental experiments to provide background noise for attackers and defenders.

Target Simulation

As understanding attackers is a big part of research for cyber security, tools for testing them are important. While it is possible to perform studies on real cyber incidents, the number and frequency of them is still relatively low and may not provide definitive answers to open questions. Target simulation capabilities are designed to provide a controlled way of stimulating attackers for study. Target simulation tools (aka honeypots) provide a way to create vulnerable targets to entice attackers into exploiting them. These tools are often named on some variation of honey, based on the concept of honey trapping, which is a spy technique of using a physically enticing target to lure assets into divulging secrets. Target simulation tools often provide extra sensoring to monitor the actions of attackers. There are target simulators for host-based systems,[25] networks of honeypots,[26] browsers,[27] and, even data.[28]

Did You Know?

The term *honeypot* has a basis in the espionage world. Honeypotting or honey trapping referred to using sexual seduction to recruit assets. Cyber honeypotting techniques were named the same by using "appealing," based on easy vulnerabilities, systems to lure in attackers.

Threat Simulation

Threat simulation attempts to recreate threats and hazards to a system to observe and analyze how the systems or users of the system react. The suite of tools that fall under threat simulation have varied uses. Some are useful for applied studies to understand systems, others provide useful tools for experimentation, and finally others provide capabilities to support validation of solutions. There are multiple categories of threat simulation tools that are useful in modeling the different tasks and behaviors of an attacker. Failure simulators, vulnerability scanners, and exploit testing platforms are discussed in this section but refer to Chapter 14, Dealing with the Adversary, for a more in-depth discussion on modeling attackers.

- *Failure simulators*: **Failure simulators** simulate hazards to a system or faults in a system. Failure simulators degrade or impede the performance of some part of the test system. This provides ability to test and evaluate remedial actions and robustness measures. Netflix created a suite of tools to test cloud infrastructure's resilience to failure.[29] Other tools provide mechanisms to test degraded network conditions.[30] Finally, there are a special type of failure simulator, called fuzzers, that use protocols or APIs improperly to test for robustness of solutions in receiving bad input. Fuzzers are a common tool for studying the security of software implementations looking for possible vulnerabilities. Fuzzers come in all types of flavors; fuzzer frameworks,[31,32] file format fuzzers,[33,34] protocol fuzzers,[35,36] and application fuzzers.[37,38,39]

- *Vulnerability scanners*: **Vulnerability scanners** are tools that search for software and service signatures to determine if they have vulnerabilities. Vulnerability scanners simulate the process of attackers in finding system vulnerabilities for exploitation. However, it is important to note that vulnerability scanners are often designed as audit tools and are not designed to accurately model the stealth tactics of attackers. The results of these tools can provide a reasonable collection of knowledge that would be gained by an attacker. There are open source[40] and commercial[41] vulnerability scanner tools.
- *Exploit testing platforms*: **Exploit testing platforms** provide packages and a framework to execute exploits against known vulnerabilities. Exploit testing platforms provide a reasonable means of simulating the capabilities of attackers for performing studies or experiments. However, by the time exploits are added to exploit test platforms, they are generally far from being zero days anymore. Therefore, exploit kits are often not a good fit for modeling cutting-edge attacker tactics. Metasploit[42] is a well-known open source exploit framework, which has spawned some add-ons.[43,44] Canvas[45] and Core Impact[46] are the major commercial exploit frameworks.

Did You Know?

Zero days are vulnerabilities that are known by attackers, but unknown by the public or developers. Their term *zero day* originated in the warez community, which were a group of people that shared pirated software. Zero day warez software referred to software that was as yet unreleased to the public. The warez and hacker communities were cross-pollinated, which led to the term being applied to vulnerabilities.

General Simulation

The rest of the simulation tools we have discussed have been specifically focused on cyber space and security. However, there are a plethora of general purpose and domain-specific simulators that could be useful depending on your research topic. Engineers use MATLAB and Simulink to model physical processes. Modelica is an integrated modeling platform for modeling many different processes. There are tools such as Portico,[47] Ptolemy,[48] and FNCS[49] that are designed to integrate and bridge different simulators of different types. Pick the right simulator for the research you are doing. Just because some simulators are not designed for cyber security does not mean they are not the best fit to answer your research question.

> Read full chapter

Rapid Control Prototyping (RCP) for a Motion System1

George Ellis, in Control System Design Guide (Fourth Edition), 2012

19.1 Why Use RCP?

There are at least two ways to use RCP and the benefits depend on which is selected:

1. Improving and validating the modelIn most industrial projects, it is sufficient to have a well-functioning physical system; it is usually unnecessary to have a highly-detailed model. However there are many exceptions, including:•When there is a desire to improve a complex plant, but it is not clear what characteristics should be modified. Having a detailed, accurate model can point to the modifications that are likely to benefit system performance; without a detailed model, making changes to the plant can be hit-or-miss.•The project may need Hardware in the loop (HIL) (Section 13.2.3) to use for product testing. By definition, HIL requires a detailed, accurate model of the plant, power converter, and feedback devices.•For high-reliability applications such as aerospace and defense, it is often required that phenomena seen in the physical plant be thoroughly understood, and that generally implies that the modeled and physical systems should show nearly identical results.•For research in control systems, it is often important to validate the model to demonstrate that the solution will work on a broad set of hardware. Showing a method worked on a particular plant is insufficient when proposing a general solution.
2. Giving access to the physical components to replace the modelWhen developing control laws, a great deal of experimentation and validation are usually needed on the physical system. Here, the model may be the starting point for development since initial development is much more efficient on simulated systems. After initial deployment, performance on the physical hardware is often the dominant concern and the RCP system can replace the simulated system as the primary development environment.

19.1.1 Using RCP to Improve and Validate the Model

Understanding precisely how the physical system behaves (which is to say, having a detailed and accurate model) is typically an iterative process:

A. Start with the model as well as it is known,

B. Execute the control laws on that model and on the physical system,

C. Compare results, and,

D. If there are significant differences, improve the model and return to Step B.

This process is repeated until the results of the model and the physical system match sufficiently well. Executing this process without RCP implies comparing a fully-simulated system with a physical system. This requires simultaneously changing over both the control laws and the power converter/plant/feedback sensor models. When the results of the simulated and physical systems differ, it is often unclear whether those differences arose from inaccuracy in the model of the physical components or from an error in the conversion of the control laws. RCP allows use of the same control laws in both cases, thus isolating most problems to the models of the physical components.

The process of developing an accurate model is usually iterative because it is impractical to include every known effect of every component in the system. Power converters, motors, loads, and feedback devices all have complex linear and nonlinear effects. For example, many motor feedback sensors have a compliant mechanical coupling to the motor shaft, which is usually difficult to characterize; most of the time, its effect is insignificant but on rare occasions it can be the dominant performance limit. How do you know at the outset whether you should invest the time to model this effect? Also, friction has many components (Section 12.4.5) and unless a given component is known to be significant, the effort to accurately model it can be excessive (especially the Dahl and Stribeck effects). So, modeling normally proceeds by first taking the primary effects into account and then adding secondary effects as experience and experimentation show necessity.

19.1.2 Use RCP to Give Access to the Physical Components and Replace the Model

Most control systems are based on standard products using predetermined control laws such as PI velocity loops. However, some systems require specialized control laws to improve performance. For example, you may need to compensate for a particular nonlinear behavior that was not anticipated by the vendor of a standard product. Or you may need to execute an unusual filter or use a novel control law not available on standard products.

Most designers start development of custom control laws in full simulation and then port those laws to a physical system. When the control laws are executed on the physical system, it is common that the results do not match the model at the outset. Typically, a tedious process ensues as the designer attempts to resolve the differences. As in the previous section, without RCP both the control laws and the plant/power converter/feedback sensor shift from simulation to physical hardware at

once so it is unclear whether the root cause of a problem might come from an error in the control law conversion or an error in the model of one of the components. Using RCP, the control laws can be fully validated on the physical plant before they are transferred to an embedded controller.

RCP provides a broad set of design tools: easy access to time-domain plots and metering of all the internal signals of the control laws, versatile Bode plot generation tools, and graphical languages to design control laws. The graphical languages provide standard control laws (like PI and PID), a wide range of filters, and a varied assortment of linear and nonlinear effects to allow designers to piece together a great breadth of control laws. The algorithms are usually executed in floating point and so are easy to drop into a system without the saturation, rollover, and quantization problems so often seen with the integer-math implementations common in embedded systems. Taken together, access to these design tools can speed the time to try new ideas and validate operation. Only RCP gives access to these design tools while executing on the physical hardware.

Of course, the control laws may ultimately have to be converted to an embedded controller since embedded controllers are typically smaller, less expensive, and more reliable than RCP systems. At that point, the designer will have to deal with complex issues resulting from the control law conversion, especially if the embedded controller relies on integer math. However, at that point the designer will be confident that the control laws work on the physical system so the great majority of issues can be isolated to the conversion of the control law to run on the embedded controller.

The remainder of this chapter will demonstrate the initial steps to design an RCP motion-control system. In Section 19.2, a servo system with a rigidly-coupled load will be converted from simulation to RCP; in Section 19.3, a compliant coupling will be substituted and the process repeated.

> Read full chapter

Deep learning for vision-based navigation in autonomous drone racing

Huy Xuan Pham, ... Erdal Kayacan, in Deep Learning for Robot Perception and Cognition, 2022

15.4.1 Simulation environments for autonomous drone racing

Simulations have long been a valuable tool for developing robotic vehicles. It allows engineers to identify faults early in the development process and lets scien-

tists quickly prototype and demonstrate their ideas without the risk of destroying potentially expensive hardware. While simulation systems have various benefits, many researchers see the results generated in simulation with skepticism, as any simulation system is an abstraction of reality and will vary from reality on some scale. Despite skepticism about simulation results, several trends have emerged that have driven the research community to develop better simulation systems of necessity in recent years. A major driving trend toward realistic simulators originates from the emergence of data-driven algorithmic methods in robotics, for instance, based on machine learning that requires extensive data, or RL that requires a safe learning environment to generate experiences. Simulation systems provide not only vast amounts of data but also the corresponding labels for training, which makes it ideal for these data-driven methods. This driving trend has created a critical need to develop better, more realistic simulation systems. The ideal simulator has three main features:

1. Fast collection of large amounts of data with limited time and computations.
2. Physically accurate to represent the dynamics of the real world with high fidelity.
3. Photorealistic to minimize the discrepancy between simulations and real sensor observations.

These objectives are generally conflicting in nature: the higher accuracy in a simulation, the more computational power is needed to provide this accuracy, hence a slower simulator. Therefore achieving all those objectives in a single uniform simulator is challenging. The three simulation tools that have tried to balance these objectives in a UAV simulator are FlightGoggles [87], AirSim [86], and Flightmare [88]. These simulation tools use modern 3D game engines such as the unity or unreal engine to produce high-quality, photorealistic images in real-time.

15.4.1.1 AirSim

In 2017, Microsoft released the first version of aerial informatics and robotics simulation (AirSim). AirSim is an open-source photorealistic simulator for autonomous vehicles built on unreal engine. A set of application programming interfaces (API), written in either C++ or Python, allows communication with the vehicle whether it is observing the sensor streams, collision, vehicle state, or sending commands to the vehicle. In addition, AirSim offers an interface to configure multiple vehicle models for quadrotors and supports hardware-in-the-loop as well as software-in-the-loop with flight controllers such as PX4.

AirSim differs from the other two simulators, as the dynamic model of the vehicle is simulated using NVIDIAs physics engine PhysX, a popular physics engine used by the large majority of today's video games. Despite the popularity in the gaming

industry, PhysX is not specialized for quadrotors, and it is tightly coupled with the rendering engine to simulate environment dynamics. AirSim achieves some of the most realistic images, but because of this strict connection between rendering and physics simulation, AirSim can achieve only limited simulation speeds. In Fig. 15.16, examples of the photorealistic information collected via AirSim are presented, providing RGB, segmented, and depth information.

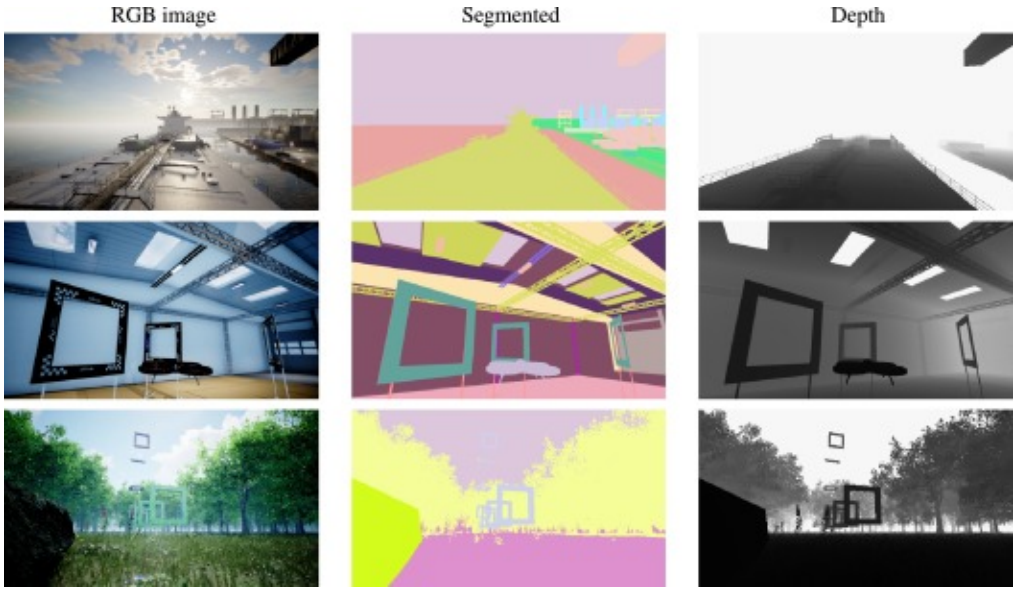

Figure 15.16. RGB, segmented and depth information extracted from three different environments using AirSim. The three environments are created using unreal engine and are showcasing; harbor (top row), drone lab (middle row), and outdoors (bottom row).

15.4.1.2 FlightGoggles

In 2019, FlightGoggles is developed at Massachusetts Institute of Technology as an open-source photorealistic sensor simulator for perception-driven robotic vehicles. The contribution from FlightGoggles consists of two separate components. First, decoupling the photorealistic rendering engine from the dynamics modeling gives the user the flexibility to choose the dynamic models' complexity. Lowering the complexity allows the rendering engine more time to generate higher quality perception data at the cost of model accuracy, whereas by increasing the complexity, more resources are allocated to accurately modeling. Second, providing an interface with real-world vehicles and actors in a motion capture system for testing vehicle-in-the-loop and human-in-the-loop. Using the motion capture system is very useful for rendering camera images given trajectories and inertial measurements from flying vehicles in the real-world, in which the collected data set is used for testing vision-based algorithms. Being able to perform vehicle-in-the-loop experiments with photorealistic sensor simulation facilitates novel research directions involving, for example, fast and agile autonomous flight in obstacle-rich environments, safe human interaction.

15.4.1.3 Flightmare

In 2020, University of Zurich robotics and perception group released the first version of Flightmare: a flexible quadrotor simulator. Flightmare shares the same motivation as FlightGoggles by decoupling the dynamics modeling from the photorealistic rendering engine to gain the flexibility to choose how much time we want to simulate an accurate model compared to gathering fast data. While not incorporating the motion capture system, Flightmare provides interfaces to the famous robotics simulator Gazebo along with different high-performance physics engines. Flightmare can simulate several hundreds of agents in parallel by decoupling the rendering module from the physics engine. This is useful for multidrone applications and enables extremely fast data collection and training, which is crucial for developing DRL applications. To facilitate RL even further, Flightmare provides a standard wrapper (OpenAI Gym), together with popular OpenAI baselines for state-of-the-art RL algorithms. Lastly, Flightmare contributes with a sizeable multimodal sensor suite, providing: IMU, RGB, segmentation, depth, and an API to extract the full 3D information of the environment in the form of a point cloud.

This section introduces some of the high-quality perception simulators on the market. Table 15.3 summarizes the main differences of the presented simulators but choosing the right one depending on the problem and what data is essential for the project. For example, if high-quality photorealistic images are the priority of the project, AirSim might be best, but if an evaluation of the quadrotor's performance is needed, maybe FlightGoggles with its vehicle-in-the-loop is the right choice. These simulation tools are being improved and extended on a daily basis, so no one can predict the right simulator to use in the future, but hopefully this section has shone some light on how simulators can be a useful tool for training, testing, and prototype and how valuable it is for autonomous vehicles.

Table 15.3. Simulator feature comparison.

Simulator	Rendering	Dynamics	Sensor suite	Motion capture	RL API	Vehicles
AirSim	Unreal Engine 4	PhysX	IMU, RGB, Depth, Seg	No	No	Multiple
FlightGoggles	Unity	Flexible	IMU, RGB	Yes	No	Single
Flightmare	Unity	Flexible	IMU, RGB, Depth, Seg	No	Yes	Multiple

> Read full chapter

Multi-paradigm modelling and co-simulation in prototyping a cyber-physical production system

Mihai Neghină, ... Ken Pierce, in Multi-Paradigm Modelling Approaches for Cyber-physical Systems, 2021

7.3 Techniques

This section introduces the main technologies used for building the initial models, including the INTO-CPS co-simulation technology and the way of generating Discrete-Event First (DE-first) models in Overture [10]. This section also presents shortcomings in directly applying the workflow of INTO-CPS and the motivation for an alternative approach.

7.3.1 The INTO-CPS technology

The INTO-CPS co-simulation technology is a collection of tools and methods that have been linked to form a tool chain for model-based design of CPSs. Rather than suppressing the diversity of the software, control and mechatronics formalisms necessary in the design of CPSs by requiring all disciplines adopt the same general-purpose notation, INTO-CPS embraces this diversity by integrating them at a semantic level [11–15], allowing engineers to collaborate using familiar modelling techniques and methods. The overall workflow and services from the tool chain used in this project are illustrated in Fig. 7.4.

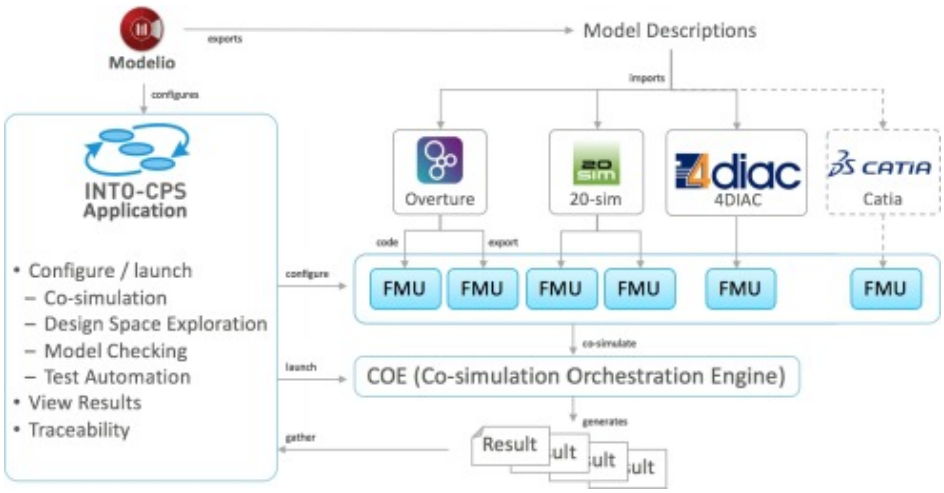

Figure 7.4. The INTO-CPS tool chain used in this project.

The INTO-CPS tool chain is centred around co-simulation of heterogeneous models using the Functional Mock-up Interface (FMI) standard [16] which allows models from different tools and formalisms to be packaged as Functional Mock-up Units

(FMUs). Such models can be combined and analysed through co-simulation, with each FMU acting as an independent simulation unit and having a model description that includes its interface. FMUs can also be provided as a black-boxes to protect intellectual property (IP) contained in the details of the model.

INTO-CPS includes a co-simulation engine, called Maestro [17], that fully implements version 2.0 of the FMI standard and has been successfully tested with over 30 tools [16]. Around this co-simulation core, the INTO-CPS tool chain links additional tools to support model-based design throughout development. A Systems Modelling Language (SysML) profile is provided and supported by the Modelio tool [18], allowing FMU model descriptions to be captured and linked to requirements. The model descriptions can capture both physical and cyber parts of the system, enabling both Hardware-in-the-Loop (HiL) and Software-in-the-Loop (SiL) simulation, and can be exported for use in modelling tools [6]. The profile also allows these descriptions of FMUs to configure co-simulations and other forms of analysis supported by INTO-CPS.

Specific support for importing model descriptions and producing skeleton models is provided by tools in the INTO-CPS tool chain: Overture [10], supporting DE modelling; 20-sim [19] and OpenModelica [20], both supporting CT modelling. These tools also guarantee export of FMUs, which can then be used in a co-simulation with Maestro [17]. FMU export is included in an increasing number of industry tools. During the heterogeneous co-simulation in the later stages of our experiment we used 4DIAC [21], an open-source tool for development of industrial control systems, and CATIA [22], an industry-standard software suite for computer-aided design and manufacturing. Both are shown in their respective positions within the INTO-CPS tool chain in Fig. 7.4.

7.3.2 Initial models

The approach of directly generating FMI model descriptions from SysML profiles and importing them into dedicated modelling tools, with the expectation that generated FMUs would seamlessly combine in the co-simulation, can be prone to failure. While allowing different teams to work on constituent models separately, it requires FMUs from all teams to be available before integration testing through co-simulation. Any delay in the generation of any component leads to further delays in the co-simulation and to the late discovery of problems. Similarly, if there is a single team producing all FMUs sequentially, in their full complexity, the co-simulation can only begin at the end of modelling.

A potential strategy to mitigate these risks is to have each team produce quick, initial versions of FMUs as soon as they can and perform integration testing with these models. The initial models can then be updated in an iterative manner towards more

detailed models, with each team able to move at its own pace, and with previous versions providing fall-backs in the case of problems and baselines for regression testing. This approach might be more difficult in some modelling paradigms, however, where quick and simple models might not function sufficiently well for testing.

The DE-first approach adopted for the IPP4CPPS project follows the strategy of producing initial FMUs and replacing them with more detailed models as they become available. Rather than using each individual formalism, a single DE formalism such as Overture [10] can be used for all initial models, thus quickly generating the abstract model of the whole system. Sketching out the behaviour of constituent models allows the early testing of assumptions at the beginning of the process. A DE formalism should be selected because these are designed to capture abstract and logical behaviours, often described in terms of interfaces, and therefore are well-suited to this task [23].

7.3.3 Discrete-event first with VDM-RT/Overture

Applying a DE-first approach to an FMI setting using Overture [10] follows the principles of the Vienna Development Method (VDM) [24,25], a set of modelling techniques successfully applied in both research and industrial application. The initial VDM Real-Time (VDM-RT) [7] project contains a class for each FMU, with port objects corresponding to the interface given in the model description, a main (system) class that instantiates appropriate port objects and instances of each FMU class to which it passes the ports and a world class that provides a method as an entry point for simulation by starting the threads of the FMU objects and blocks until simulation is complete.

Fig. 7.5 provides class and object diagrams and shows such a set-up using two constituent models, FMU1 and FMU2. Such a model can be simulated within Overture to see how the FMUs behave and interact. Once sufficient confidence in these initial models is gained, they can be exported individually as FMUs and integrated in a co-simulation.

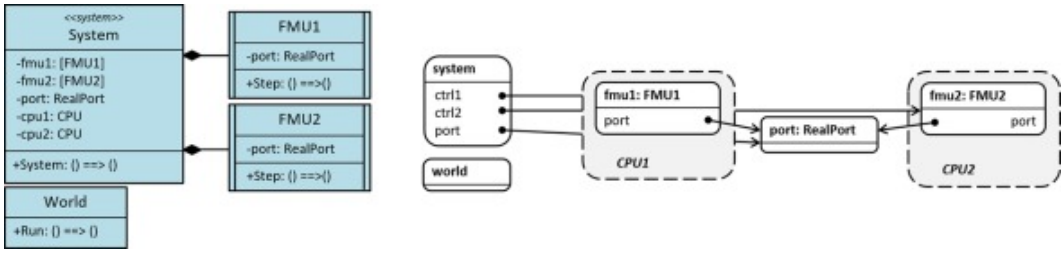

Figure 7.5. Class diagram showing two simplified FMU classes created within a single VDM-RT project, and an object diagram showing them being instantiated as a test.

The Overture FMI plug-in can then be used to export an FMU from each individual project unit, these can then be combined in a co-simulation. These FMUs can be revised if problems are found, then replaced with higher-fidelity models. The models could be retained for later use and as a fall-back in case of future problems in integration.

> Read full chapter