Imperial College London,
Department of Computing

A Framework for Analysing Cyber Physical Systems in 3D Virtual Environments

Fergus Leahy

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, October 2018

Abstract

Text of the Abstract.

Acknowledgements

I would like to express (whatever feelings I have) to:

- My supervisor
- My second supervisor
- Other researchers
- My family and friends

Dedication

Dedication here.

‘Quote text here.’

Guy Quoted

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Cyber Physical systems bridge the boundary between the virtual and real world, the cyber and physical, respectively, in which the virtual directly impacts the real physical world, and vice-versa. CPS rarely consist of a singular device, but are instead made up of myriads of different devices, each of which perform actions towards one or more collective goals, distributed across the physical environment. As these systems become relied upon for providing more critical services within our physical environment, more focus is needed on ensuring these systems perform correctly once deployed in their target environment.

1.1 Motivation

Why is it important to have a better simulation tool for CPS testing and analysis.

1.2 Problem

Testing CPS is carried out in two main phases, simulation and pre-deployment.

Simulations are used to develop and iterate algorithms quickly and efficiently, without the overheads, time and cost, associated with programming and deploying physical devices. However, the simulators lack the ability to simulate the physical aspects of the CPS. Developers instead rely upon recorded data, manual input or fabricated scripts to act as the environment and close-the-loop within the simulation. Recorded data provides a realistic input from a singular perspective, but the data is fixed and can't be adapted to suit different sizes or layouts of a CPS. Manual input and scripts provide some more flexibility but only at a limited scale, still requiring considerable work to change and adapt to different sizes or layouts. Similarly,

these also depend on a developer's model of the environment, which may not be an accurate representation of what can exist or occur within the real counterpart.

On the other hand, pre-deployment tests can be carried out once an application is deemed stable, and full integration and reliability tests are needed, providing a realistic mini-deployment to test the interactions between the cyber and phyiscal components of the system within. However, pre-deployments are expensive and time-consuming to run, due to device acquisition and physical deployment, respectively. Pre-deployments are also limited in what can be tested, due to physical or cost constraints, and health and safety.

Describe the problem upfront for CPS expert audience:

- Must test CPS using pre-deployments and simulation.
- How to simulate environment, people, phenomena? Accurately? Virtual world?
- How can we analyse these systems in-place, in real-time, in an intuitive way?
- Can we validate and compare testing in the virtual world using real deployments?
- How can we transfer these tools and apply them in the real-world? (AR).

1.3 Objectives

How does this thesis aim to resolve the problem outlined above?

1.4 Contributions

Discuss our outputs?

- 3D CPS Simulation platform
- Visual Diffing tools for analysing 3D simulations in real-time
- Visual Diffing tools transferred into AR for use in real-world

We make the following contributions:

- Framework - A co-simulator framework for the development and testing of indoor Cyber-Physical systems **??**. The framework provides the scaffolding for performing a collection of co-simulations utilising different tools for real-time simulation analysis.

- 3D Simulator - A novel 3D CPS co-simulator for deploying CPS within a 3D virtual world utilising realistic physics, lighting and human mobility.
- Visual Diffing toolset - A novel simulation analytics tool-kit for comparing two simulations in real-time from within the 3D virtual environment.

1.5 Statement of Originality

Statement here.

1.6 Publications

Publications here.

Chapter 2

Background (20-30Pages)

2.1 Introduction

Describe the state-of-the-art tools and techniques for debugging, testing and analysis of CPS systems.

- Deep analysis and critical evaluation of key related work.
- Cover breadth lightly about relevant work.
- Show critical thinking and **GAPS**.
- Discuss related info important to understanding thesis.

10-20pages

2.2 Testing Cyber-Physical Systems

2.2.1 Pre-deployments

Discuss using pre-deployments to test a CPS.

- Real devices with real environment (context, light, radio, etc)
- Time consuming, impracticle, infrequent phenomena, no debugging

2.2.2 Test-beds

Discuss the use of test-beds for testing CPS, including the use of federated test-beds 3-5 pages Mention the:

- benefits of debugging tools (gdb)
- Drawbacks of restricted deployment, fake environment

2.2.3 Simulation

Discuss the use of simulation in testing Cyber-Physical systems as well as sensor networks and robotics. Mention the:

- benefits of using gdb, speed and scalability
- lack of environment simulation

Discuss the use of checkpointing 5 pages

Contiki - Cooja

2.2.4 3D simulation - Robotics

2.2.5 Mixed-simulation

Combination of virtual and real devices, for expanding a network or mapping real hardware (radios, sensors) to virtual nodes. 2 pages

2.3 Analysing Cyber-Physical Systems

Discuss the techniques used for analysing CPS deployments with references to existing work.

2.3.1 Record and Replay

2.3.2 Tracing

2.3.3 Log Analysis

2.3.4 Visualisation

2.3.5 Model Checking

Use model checking to generate scenarios based on parameters for **placement, power, number, design etc.**

2.4 Game Engines

3D games engines, such as Blender[1], Unity 3D[3], Unreal Engine 4[4] and CryEngine[2], power many of the current generation video games, providing players with game worlds filled with rich visuals, audio, physics and artificially intelligent (AI) agents.

In order to support the high cost and time consuming nature of video game development, large development studios typically create a single game engine to support fast-paced development of a wide variety of games. These game engines provide an abstract foundation to build scenarios in, incorporating flexible design tools, 3D modelling, programming tool-sets (C++, C#, etc), physics and lighting engines, networking, multi-player support and more recently virtual reality support, offering it commercially for smaller studios to bootstrap their game development.

2.4.1 Physics and Lighting

Key features of games engines, aside from the 3D modelling and graphics tools, are their support for advanced physics and lighting simulations. Typically performed by middleware, physics engines, such as PhysX[**?**] and Havok[**?**], provide real-time realistic physics including collision detection, rigid- and soft-body physics, forces and motion, fluid and particle simulation, and destruction. Using these tools, game worlds and the objects within react to the player as we would expect, such as objects falling due to gravity. Similarly, games engines also provide advanced lighting, enabling the use of both static and mobile lighting, with dynamic shadows, occlusion, reflection and refraction. Lighting is key to bringing virtual scenes to life, by illuminating spaces, guiding viewers attention and creating natural divisions between areas.

2.4.2 Update and Render cycle

In each time period, known as a tick, the game engine renders the 3D world using a two step method, update and render. On a tick, first, the state of the world is updated, allowing code to run for making AI decisions, physics simulations and updating character positions based on user input; second, these updates then applied to the world and rendered to the screen, known as a frame.

The rate at which the 3D game engine can render to the screen is recorded in frames-per-second, with higher frame-rates providing smoother video with more responsive physics simulations and interactions, due to code running more frequently. However, even with decreased frame-rates (<60 FPS) the game still runs in real-time, albeit with the resulting video and simulations becoming jittery and input lag becomes significantly noticeable. Therefore, because code can only run once per tick, reducing the overheads in both simulation algorithms and rendering is paramount to increasing the frame-rate, thus improving the overall simulation. To a certain degree, high performance graphics cards can also significantly increase the speed of the render component of the tick (>120 FPS).

2.4.3 Continuous operations

Whilst game code typically runs within the update and render thread described previously, some operations, such as long running or externally communicating ones, aren't suited to this approach. To solve this issue, it is also possible to run operations in continuous running threads outside of the main game thread (update and render). This allows the thread to run operations in parallel to the game and at a faster or slower rate than when tied to the limited frame-rate; this enables processes to process information independent of the frame-rate and provide more responsive feedback to external systems.

2.5 Virtual & Augmented Reality

2.6 Summary

Purpose of this section is to show:

- state of the art
- critical thinking

- gap analysis

Chapter 3

A Framework for Simulating and Analysing Cyber-Physical Systems

Despite the distributed and physical nature of cyber-physical systems and the Internet of Things, tools and techniques to support the needs of their development have not evolved to support efficient and reliable testing in the context of the deployment environment. Many tools exist which support individual aspects of testing and development, however, each one is siloed from the rest, limiting the overall ability and coverage of testing.

This chapter presents a CPS co-simulation framework designed to close the cyber-physical loop in CPS testing. Treating the physical environment as a first class citizen, the framework makes it possible for developers to simulate and test CPS within a 3D virtual environment, pre-deployment. A CPS interacts with and is directly affected by its target environment and inhabitants within it. The framework makes it easier for developers to test and analyse CPS which rely upon human mobility and device placement.

The following sections in this chapter outline a set of requirements **??** for the framework, a distributed design for building a co-simulation platform supporting plug-and-play of multiple tools, including a sensor network simulator and 3D game engine. Finally, we describe several case studies which are used to demonstrate and evaluate the co-simulation framework and its features.

3.1 The Ideal System

This section describes a key set of requirements that are necessary for a framework to support simulating and testing cyber-physical systems in virtual environments not only comprehensively

and accurately, but also efficiently and effectively, such that developers benefit from using such a tool over simply testing in the real world or in a sensor network simulator.

Each of these requirements focus on key aspects needed for the framework to be effective for testing cyber-physical systems, which are deeply integrated into our physical world.

3.1.1 Placement

The issue of placement in CPS is tightly coupled to provisioning, i.e., what is the minimum number of devices needed and where should these devices then be placed in order to acheive the desired coverage, reliability, or other desired metric. Understanding how many devices are needed for a specific deployment and where to place them within the environment is a difficult challenge pre-deployment. Often based on guess-work, trial-and-error and field studies, which can lead to both over- and under-provisioning, resulting in either high-cost, a lack of network reliability, insufficient or redundant sensor data, etc. Thus, virtually experimenting with placement ahead of time will help mitigate these issues before deployment.

Testing different placement strategies should be quick and efficient without the need to manually change scripts and sensor input traces; instead, the simulator should dynamically generate inputs for devices under test. Similarly, developers should be able to flexibly scale the levels of device provision, to test and understand how it will affect a deployment. Ideally, the system should support automatically testing and suggestion of optimum placement and provisioning levels, based of some selected criterion, such as cost, quality-of-service, robustness, etc.

3.1.2 Phenomena-on-demand

In order to test a CPS in the real-world, developers often need to wait for or force desired phenomena to occur in order to observe how the system reacts and whether or not it reacts correctly. However, exercising control over the real-world is a difficult and time-consuming challenge. Similarly, enlisting human participants to perform repetitive mundane tasks, requires paperwork, significant time and often money. Lastly, some scenarios are simply not feasible to test due to their scale, a deployment environment's limited access or health and safety concerns, e.g., outdoor environments, evacuation, fire, flooding.

Ideally, the framework should provide a dynamic and flexible method to create and/or script phenomena which interact with the system on demand. Unlike traces, a scripted phenomena, such as a person walking down a corridor, only needs to be updated once to change path with the virtual world dynamically generates sensor input, whereas, multiple traces feeding sensors

simulated "detections" inputs need to be updated for each sensor device in the person's path, a time-consuming and unscalable solution.

3.1.3 Human Mobility

Existing tools support the concept of device mobility, updating a device's radio model based on its location. However, in these tools device mobility in is pre-determined; in other words, developers define mobility patterns, as a sequence of positions, manually, which the network of devices must follow, but don't react to the live cyber-physical simulation. The tools have no concept of other entities, such as people, physical objects or gravity.

Because of the lack of support for simulating human mobility, developers are required to either manually trigger sensors or create custom mobility model scripts for each test scenario, in which the script triggers button presses or motion sensors at specific time intervals. As the network grows, shrinks or the layout changes, developers need to update the model for each device; this quickly becomes unmanageable as the network and number of tests scale in size.

Ideally, the framework should support testing CPS utilising real-time dynamic human (and device) mobility, enabling simulations to support testing environments in which virtual human participants interact with and are affected by the CPS and the environment, closing the cyber-physical loop. In other words, virtual participants are able to autonomously navigate an environment, based on flexible pre-assigned behaviours; including simple actions, such as patrol this corridor or walk to location A; or more complex behaviours, such as follow or avoid person X, wait on an event before moving. Whilst target locations may be pre-defined, routes may change in real-time based on obstacles or other behaviours. By combining behaviours, participants can dynamically generate phenomena for the CPS to sense and react to, in turn forcing the participants to react, and so on.

Using such a system, when simulating a fire evacuation scenario, participants dynamically react to fire avoiding it as it spreads throughout the environment, with the CPS able to sense, alert and direct evacuees towards safe exits.

On top of this, it should also be possible to provide complex mobility and crowd behaviour, such that individual participants can be set to join and leave crowds of participants, moving as a unit through an environment.

3.1.4 Time Control

In real-world deployments developers have no control over time and have to analyse events as they happen or rely on logs and video to record and view post-test. Simulation approaches

enable pausing and slowing time, giving developers more time to analyse snapshots or small windows of time in a test, however, recording still utilises log-based approaches, providing only a limited snapshot. Both approaches provide limited views into a test, if events were not recorded, via logs, video or otherwise, they are lost and the test must be re-run. Recording everything would enable a simulator to create a full reconstruction and allow developers to master time-control, moving forwards and backwards through a simulation to view its state at any point in time. Within a 3D environment, this would also allow developers to observe the context in which particular phenomena occur, enabling in-depth and contextual analysis.

Naturally, the system should support recording arbitrarily long, such as day- or even week-long, simulations, enabling longitudinal testing of CPS applications in real-time or faster. The system must also provide the means to quickly scrobble through recordings, based on time or checkpoints, generated based on points of interest.

3.1.5 Distributed component-based simulation

The co-simulator should be discretised into components with clear interfaces; such that the architecture allows for distributing components across multiple machines. Similarly, the architecture should support for allowing virtual components to be swapped out for other components, or their real counterparts e.g., swapping out a set of virtual simulated nodes for a set of real deployed hardware nodes.

3.1.6 Visualisation (Visual Diffing, Analytics)

To complement log output, existing CPS simulators provide basic visualisations, such as 2-dimensional network maps and transmission timelines, giving developers more insight into the network component in real-time. Many of these visualisations are transient and can only be viewed once as the simulation is on going. Thus, when trying to compare two separate simulation runs, developers still need to resort to post-simulation log analysis using external tools to manually process the log output, without the context of the simulation visualisation.

When simulating in the context of a 3D virtual world, visualisations must add value to already busy simulations, with the aim to reduce the visual noise, not add to it. Thus, visualisations should not simply be added on top of the 3-dimensional virtual world, instead they should seamlessly integrate into the virtual environment to provide intuitive visual feedback, indicating points of interest, such that they further enhance a developers analytical tool-kit and understanding of an on-going simulation.

Ideally, visualisations should help developers to spot and analyse differences between multiple runs in real-time, without the need the need to leave the tool to perform post-simulation data analysis using manual tools. These visual differencing tools, would allow developers to specify events of interest to watch, which the simulator then visually flags to the user when it happens, e.g., a node glows red when it's power levels drop below a certain point, or differ from a previous simulation run by x%. Similarly, visualisations should also support longitudinal studies of systems.

3.1.7 Virtual Sensors and Actuators

Sensor and actuator devices have both a hardware and software representation, thus, a framework must reflect both of these components within the simulated world. The CPS simulator runs the device program code, simulating the CPU, network and other virtual components, whilst within the game engine the device has a physical form with sensors and actuators that can perceive and interact with the virtual world.

This will enable a co-simulation approach in which simulated hardware devices can interact directly with the 3D virtual environment, and not just with faked or simulated input from traces or user input. The framework will provide developers with flexible and dynamic environmental interaction between the CPS and the environment, closing the loop within a simulation.

Visually, sensor hardware can be modelled at varying degrees of accuracy, depending on the requirements, e.g., virtual devices can be larger in size compared to real counter-parts, enabling easier location and observation. Similarly, sensing can be simulated to varying degrees of accuracy, again depending on the needs and simulation power available.

3.1.8 Direct control

The framework should allow direct control of three key aspects of the simulation: entities in the world, such as environmental attributes and physical devices or sensors; virtual people in the 3D world, enabling the developer to inhabit the world vicariously through a virtual person, interacting and responding to the world as if it were the real world; lastly, the view, allowing the developer to view the world from any angle at any point in time, including pausing the world mid-simulation and moving around, providing a better view of a point of interest.

3.1.9 Write Once, Test Everywhere

Using a single language for both testing and deployment, in the real- and virtual-world, significantly reduces the time and effort for transferring between the different environments and ensures a consistent code-base, with the hope for consistent performance and output. Similarly, using an existing language reduces barrier to entry for use of a new tool and eliminates the opportunity for translation bugs being introduced when transitioning between test and deployment, whether they be real or virtual environments.

3.1.10 Performance

The system should support real-time operation so that simulations run as fast as, if not faster than their real counterparts, enabling quicker deploy-test-debug cycles. Similarly, it needs to be responsive to real-time direct interaction by developers, such that developers can observer, interact with and change ongoing simulations. Thus, in order for the framework to run in real-time, each of the components must also run accordingly. In order for the sensor network simulator to run in real-time, the 3D game engine must respond to all requests in real-time (sensor reads), such that the virtual hardware is in-discernible from the real counterpart.

3.1.11 Synchronisation

Synchronisation, between the various components, including CPS simulator and 3D game engine clocks, ensures time sensitive events which rely on a global notion of time and delays occur correctly. Whilst real-time operation is also a requirement for all sub-systems, due to the differing simulation methods which aren't 100% accurate and the intricacies of OS level scheduling, synchronisation is necessary to ensure that the individual sub-system's clocks stay in sync, within reasonable bounds. Similarly, communications between the sub-systems must be of low enough latency such that no additional delays are introduced. Typical applications are designed with predictable or known delays, such as opening a door after a motion event.

3.1.12 Test cases

The system should provide support for automated testing, to observe and assert conditions about not only the CPS, but also the virtual world environment, akin to unit-testing available in traditional application development. Assertions would provide developers with warnings about phenomena that have occurred on a particular node, area, or within entire environment, such as ensuring a light's duty cycle doesn't exceed a set limit, checking a group of nodes are

reporting similar information, or checking for network partitions once a node has moved out of an area.

3.1.13 Modular environment building

Designing 3D environments from scratch can be a slow and difficult process for non-expert 3D designers such as developers. Thus, in order to enable fast set-up, the system must support re-use and modification of existing pre-built environments and provide modular building blocks for building environments e.g., walls, doors, rooms, sensors, lights, agents.

3.1.14 Scalability

The scale of cyber physical systems can range significantly between self-contained deployments within a single room to ones which cover entire buildings, streets or even cities. Deployments could be internal, external or a mix, each of which may effect design decisions and issues that may arise. A simulation framework needs to be capable of supporting from 10s to 100s of devices. Similarly, the simulator must also be able to support rich environments, including complex layouts, large numbers of people to inhabit the world and environmental effects.

3.1.15 Extensibility

The framework design should facilitate seamless integration of new tools to supplement or replace the existing ones. This allows future developers to build upon and improve the co-simulation framework, its output and experience through the use of additional tools, including simulation tools, analytical tools, model checkers or real test-beds.

3.2 Framework

The current state-of-the-art for cyber-physical systems focuses on the network and hardware simulation components of testing, however, many other tools and platforms exist that could support other aspects of it, such as environment, physics, analysis without the need to directly extend or integrate these features directly within any of the existing tools. The goal of this framework is to provide a stable and extensible platform into which a variety of tools can be integrated into to support the simulation, testing, modelling and analysis of cyber-physical systems.

This section proposes an architecture for designing a distributed framework supporting the co-simulation and analysis of cyber-physical systems using multiple different tools. This will enable information sharing between tools, enabling these tools to serve their purpose and close the loop, e.g., 3D simulator updates positions of devices based on physical interaction with the virtual world, this is then reflected in the network simulator which affects the radio transmission range and interference.

3.2.1 Architecture

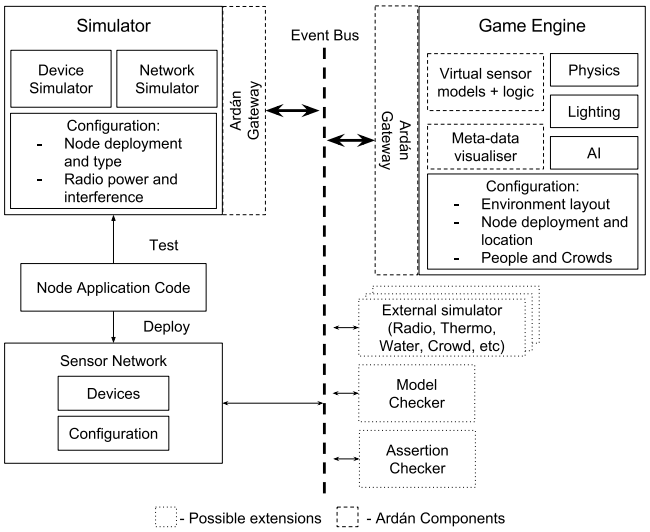


Figure 3.1: Ardán architecture

Thus, rather than integrate the systems directly, our approach uses a pub-sub event bus and common schema to describe information passed between the different tools. This approach reduces the tight coupling between tools, allowing individual tools to be removed or new ones added with minimal configuration. We envision the use of tools such as model checking, statistical analysis, unit-testing and advanced simulation for radio and environmental properties.

3.2.2 Communication Model

Discuss the use of a pub/sub for distributing simulation events and interactions between co-simulators.

3.2.3 CPS Simulation

Discuss the use of the Cooja simulator to simulate C based sensor nodes using both emulation and simulation.

3.2.4 Game Engine - Environment Simulation

Discuss the use of a modern game engine, Unreal Engine 4, to model and simulate a physical environment, including people.

3.2.5 Time

Describe the model of time and how different components coordinate.

Chapter 4

Case Study

4.1 Introduction

Describe the case study which will be used throughout the thesis to demonstrate old methods vs new methods.

4.2 The Corridor

4.2.1 Environment

4.2.2 People

4.2.3 Sensors + Actuators

4.3 Fire Evacuation and Navigation

This case study focuses on the concept of a deploying a CPS to aide in navigating a safe evacuation route during a fire alert. The CPS will take a distributed approach to determining a safe route, without the use of global maps or centralised decision nodes. Instead nodes will rely on only local layout knowledge and must determine safe routes through dissemination of locally safe paths.

4.3.1 Layout

Nodes will be placed along corridors, doorways and/or rooms equiped with fire detectors (smoke, heat, etc). Nodes will also be equiped with directional indicators to designate safe routes to an exit. Nodes can be of two types, a pathway node, or exit node. Exit nodes are placed adjacent to fire escape routes, such as stairwells, or exiting doors.

4.3.2 Invariants

- No node should direct towards fire.
- No two adjacent nodes should point towards one another.
- Exits near fire should be avoided.

4.3.3 Algorithm design

```
for(;;) {
  wait_on_event();
  if (ev == ON_FIRE) {
    status = ON_FIRE;
    broadcast(FIRE_ALERT);
  } else if (ev == FIRE_ALERT) {
      blacklist_dir(msg.src);
      status = FIRE_ALERT;
      broadcast(FIRE_ALERT);
      if (self.type == EXIT) {
        broadcast(SAFE_ROUTE);
      }
  } else if (ev == SAFE_ROUTE) {
      whitelist_dir(msg.src);
      broadcast(SAFE_ROUTE);
  }
}
```

Chapter 5

Simulating Cyber-Physical Systems in the Virtual-World (10)

Testing and understanding how the environment interacts with a sensor network and vice-versa relies on placing the devices in the target environment and waiting for/creating the desired phenomena to interact with the network. Performing tests in the real-world is time-consuming, costly and often impractical. Carrying out tests utilising human mobility also poses difficulties. Phenomena can include events such as movement of devices/objects, passive or active interaction with people (pressing buttons, triggering motion sensors), or other sensor events. Performing test deployments can be expensive, time-consuming and difficult.

5.1 Problem

Cyber-physical systems are deployed in physical environments, interacting with the surrounding physical infrastructure to provide services for nearby inhabitants, such as heating, lighting, security, fire and environmental safety. However, existing simulation tools and techniques used to develop, test and analyse such systems don't consider the physical environment, its inhabitants and their mobility.

Existing approaches for testing CPS focus heavily on accurately simulating devices, the network and power consumption, with great success [8, 7]. These tools and techniques don't support reliably, comprehensively and dynamically testing these devices in the context of their target environment and the phenomena which may occur within it. Typically, the environment data is fed into simulations using recorded sensor traces or fabricated based on a developer's testing model and needs. These traces suffer from several issues:

- static - recorded traces are taken from fixed positions, can be modified but is time consuming and un-reliable.

- reliability - fabricated traces are based on a developers model of an environment and may not accurately represent it or all of it.

- comprehensive - traces are time consuming to create and must be created ahead of time.

- scalability - updating traces when new devices are added or the scenario changes becomes unmanageable for even small networks, ¡10 devices.

Gelenbe et al.[]

In the remainder of this chapter we describe the new co-simulation platform, based on the framework described in chapter 3, which focuses on enabling CPS simulation within its target environment, utilising a dynamic virtual world.

5.2 Design

The WSN simulator (Cooja), performs all application and network simulation for the co-simulation and is configured with a set of nodes with their 3D location and application code to run. As the application code is run in real-time, any sensor- or actuator-based hardware requests, such as sensor reads and actuator commands, are forwarded to the 3D game engine to be performed in the virtual environment.

Within the Unreal Engine we modelled a 3D environment and created a several components to support deploying virtual sensor networks, including device and sensor 3D models, hardware abstractions for the virtual sensors and actuators to sense the virtual world and report back to the simulator over the network bus. Similarly within the Cooja simulator we have modified the simulated hardware to communicate to our virtual hardware in the Unreal Engine.

Using Cooja, developers are able to write code once and both test in Ardán and deploy in the real-world on the Contiki or TinyOS platform, removing difficulties associated with translating between test and deployment platforms. Using the open architecture, support for additional tools and simulation platforms can also be added, expanding the possibilities and improving the fidelity of simulations.

5.3 Phenomena-on-demand

Unlike existing approaches which utilise purely trace-fed and scripted approaches which are static and difficult to modify, this simulator enables dynamic phenomena generation based on the user input or scripted behaviour. By taking direct control of devices or people within the environment, developers can interact with and modify the simulation directly and observe how the dynamic simulation responds. This differs to trace-fed alternatives which require developers to manually change the input to sensors based on how they believe the environment has changed.

For example, two sensors are placed within a school corridor to detect the pace of people walking past; the sensors communicate and calculate the walker's pace, displaying a red light to people running. In a trace-fed scenario, developers script sensor detection inputs to the two nodes. Developers may generate a test case for different speed walkers or multiple walkers. For each test case, the developers need to manually calculate and modify each trace-feed for each sensor individually. For our approach, developers need only change the speed of a person walking and observe the effects on the system. This approach provides a much more intuitive and scalable solution to testing with large numbers of devices and as the number of input sensors increase, in which trace-fed solutions become unmaintainable.

5.4 Mobility

5.5 Communication

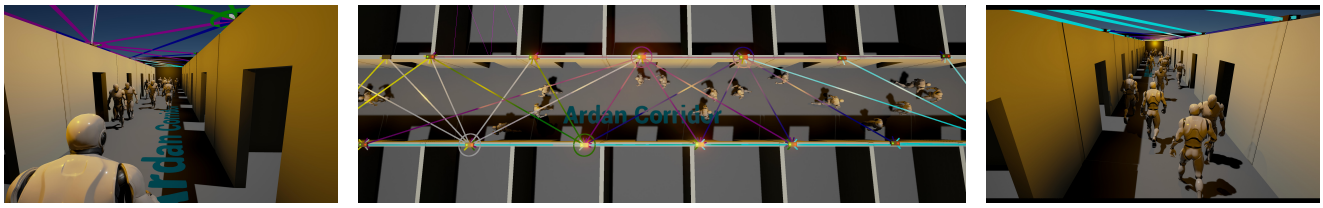
5.6 Synchronisation

5.7 Forward Time Control

5.8 Distribution

5.9 Case Study

The purpose of the following case study is to demonstrate what development of a non-trivial CPS application is like using Ardán, and how it enables developers to test and visualise different scenarios, by simply adding or moving devices or people in the simulation.



(a) Over-the-shoulder view (b) Birds-eye view (c) Camera view

Figure 5.1: 15 people walking up and down the virtual corridor, triggering motion sensors

The case study focuses on controlling the lighting within a modern day office corridor, with the goal of striking a balance between energy efficiency, effective lighting and user comfort. The ideal corridor lighting scheme should provide a pleasant lighting scheme for users of the corridor, able to adjust based on ambient light levels, gradually illuminating as they progress through it, whilst also ensuring energy is minimised by turning off or reducing the brightness of unused or infrequently used parts of the corridor.

Thus, this provides an interesting and non-trivial task, due to the many ways in which the corridor can be entered/exited or moved around within in; people can enter from the beginning, end or from a room; people can move down the length of the corridor or directly from room to room; people often also stop in the corridor, spending time talking or waiting. Similarly, understanding the ideal number and placement for devices and sensors, and how it affects applications, such as power, reliability, robustness etc. Hence, providing effective lighting schemes can prove difficult to analyse and reason without rigorous testing.

The rest of this section will demonstrate and discuss the use of Ardán to design, analyse and test for our target environment, the corridor, highlighting the features and benefits that the tool provides.

5.9.1 Corridor Design

We created a virtual corridor based on a real corridor within our building, measuring 20x 1.5 meters, with 5 doors spaced evenly on either side. Along the corridor we placed 15 nodes with lights and conical motion sensors attached to the ceiling facing the floor directly below, shown in figure **??**.

To construct the corridor, we used pre-existing models for walls, doorways and lights, making it quick to build or modify. In addition to these, we've created several new models for nodes and a variety of sensors, which can be composed together to create different sensing devices. A drag-and-drop interface is used to place nodes within the newly built 3D environment. The node model is based on a small box with 3 coloured lights, representing the typical LED outputs available on devices, such as the TelosB mote.

```
wait (event):
  if event == network:
    if msg.src + 1 == me or msg.src - 1 == me:
      turn_on_light(3000)
  if event == PIR:
    turn_on_light(5000)
    alert_neighbours()
```

Figure 5.2: Lighting algorithm psuedo-code

The next step was to create the nodes in the Cooja simulator, compiling and loading node application code. Using the IDs Cooja assigns to these nodes, the virtual representations were assigned matching IDs. This is especially important when certain applications are loaded on particular nodes, or when node IDs are used programmatically e.g., for location, routing or ordering.

5.9.2 Lighting Algorithm

For demonstrative purposes, we developed a basic lighting algorithm based on our needs described previously. The algorithm, shown as pseudo code in figure 5.2, and developed further in figure 5.3, waits for a motion detection event before illuminating its light for 5 seconds and notifying its closest neighbours. If it receives a message from a neighbour, it checks that it's adjacent, before illuminating its light for 3 seconds. The second variation attempts to build upon this algorithm, improving the lighting efficiency, using the simulator to test and experiment.

Using this as a base, we expect to test and iterate the algorithm based on our findings from our "What if?" scenarios.

5.9.3 "What if?" scenarios

When testing CPS deployments, "what if" questions about how the system will perform will naturally arise, such as "what if we move or increase/decrease the number of nodes?", or "what if there are multiple people?", or "what if we place sensors differently or use more/less sensitive ones?". Being able to quickly test and understand what happens to a system in these different scenarios is key to improving its reliability and efficiency.

In order to test our lighting application we devised several test scenarios to test both basic and complex situations for which we expect the system to perform correctly with; the complexity of a scenario increases as the number of agents in the scene increases and the pattern of movement

```
wait (event):
  if event == NETWORK and msg.dst == me:
    turn_on_light(5000)
    if msg.src + 1 == me:
      dir = FORWARD
    else if msg.src - 1 == me:
      dir = BACKWARD

  if event == PIR:
    turn_on_light(5000)
    if dir == FORWARD:
      alert_neighbour(FORWARD)
    else dir == BACKWARD:
      alert_neighbour(BACKWARD)
```

Figure 5.3: Lighting algorithm psuedo-code with direction

changes from simple start to end directions, thus becoming more difficult to visualise and debug conceptually.

Using Ardán, we are able to directly control a person in the virtual space, directing them down the corridor and observing, from multiple angles, the lighting algorithm reacting to their presence. This provides ultimate control in creating dynamic and new test scenarios, allowing developers to run around without any of the drawbacks of performing the same tests in real-life, such as fatigue, health and safety and time.

We also have the ability to adjust the number and placement of nodes within the environment, enabling us to test different configurations and determine which works best and fits our requirements.

Unlike the real world, using Ardán we are able to pause the entire simulation, giving developers more time to understand the state of the network and virtual world at a particular point in time, before stepping through or continuing the simulation. On top of this, we are also able to change our view point between the cameras placed in the virtual world or move freely about within it to fully capture and understand the state of the environment whilst the simulation and world are paused, otherwise not possible in recorded videos of a real-world deployment.

Tests:

1. One person
    - Walks from end to end
    - Walks from room to room
    - Walks, stops, walks opposite direction

2. Multiple people
    - Two agents walk from opposite ends
    - Multiple agents walk in different patterns
3. Number and Placement of nodes
    - 6 nodes, evenly spaced
    - 6 nodes, with additional sensors placed facing doors
    - 12 nodes, evenly spaced

5.10 Evaluation

To prove useful for a developer our system needs to scale to support large networks whilst ensuring synchronised behaviour at real-time or faster. To demonstrate the scalability of Ardán, we designed a case study based on managing automated lighting in a typical office corridor, illustrated in figure 5.1. Within the corridor we placed the sensor nodes and motions sensors. As people walk through a motion sensor, the relevant device is triggered and turns on its light before notifying its neighbours. When neighbours receive a notification they too turn on, providing a path of light illuminating around the walker. This task provides a non-trivial challenge for designing and testing applications that can deal with various scenarios that could occur, including multiple people walking in different directions, entering/exiting for different areas and people stopping/loitering.

The tests were run on the following spec machine: Xeon E5 1650 6Core with HT, 16GB RAM, 256GB SSD and a sufficiently powerful (MSI GeForce GTX 970) graphics card to support the game engine, otherwise, slow response times between the simulator and 3D game engine would cause the simulation to stall and drop below real-time performance.

5.10.1 Co-simulator Performance

The results in figure 5.4 show that Ardán can support up to 200 sensor nodes running at real-time with the simulator staying reliably synchronised with the game engine. Beyond this, the simulator performance degrades quickly and synchronisation is lost.

Extending further, we also performed tests on running Ardán at faster than real-time, at 200% speed. In this mode, the game engine and its physics engine match the speed of the simulator, resulting in all activity increasing in speed, as opposed to simply increasing the walking speed

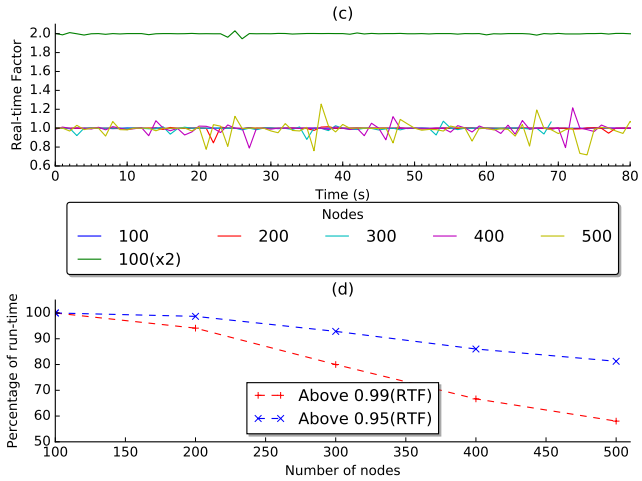


Figure 5.4: Figure (a) shows how the simulator speed fluctuates over time. Figure (b) shows the percentage of the total run time which the simulation maintains above 99% and 95% of its target speed.

of individuals. The results in figure 5.4 show that roughly half the number of nodes can be simulated in time with the game engine, with little fluctuation.

5.10.2 Faster-than-real-time Performance

5.11 Summary

Chapter 6

Visual Diffing a CPS in a Virtual World (15 Pages)

6.1 Problem

Testing a CPS in either simulation and the real-world typically involves recording data to logs, before then reading and processing (using tools or custom scripts) the logs off-line, in order to analyse, discover and diagnose issues. Simulation tools provide the possibilities to record more data due to the increased access and lack of constrained hardware, however, are still limited in how they can present it to users, still heavily relying of textual logs.

Processing log data manually is extremely time-consuming and only provides limited context given the amount of information a developer can read and process at one time. Utilising scripts or external analytical tools can provide more intuition about the data in significantly less time, however, results are provided off-line and outwith the context of the experiment.

Testing in the virtual world provides developers with access to significantly larger volumes of data about a simulation than is possible to record from the real world, including information about entities in the world and the physical environment; combined with a fully customisable and controllable 3D world, this opens up the possibilities to more advanced and visually integrated analytical tools.

However, this benefit comes at a cost. Firstly, with such large volumes of data, at the rate of gigabytes per hour, how can developers process, analyse and make use of all the data in real-time, without resorting to reading and parsing logs? Secondly, using this data, how can developers visually compare and contrast two or more simulations, to identify differences when performing A-B testing?

- Data Volume, how to compress and index in real-time or faster
- Visual Bandwidth, how do we show data - how much can we show
- How can we compare two streams in real-time

6.2 Visual Diffing

Understanding differences between two or more entities is a common but vital task across variety of domains, including text, image, voice and video analysis. System administrators or developers who need to know the differences between two text files employ the 'diff' UNIX tool[1], which highlights lines that don't match between files; Doctors and medical staff compare medical scans, often overlaying one onto the other with a backlight, to spot key differences; In sports, video replays are combined to show two or more racers simultaneously (Dartfish), or utilise high-speed multi-camera arrays to generate 3D models to discover if tennis shots are out of bounds (Hawk-eye).

Inspired by its use in other domains, visual differencing, "visual diffing" for short, enables viewers to observe the both visible and non-visible differences between two or more instances of an event incorporated directly into the visual medium itself. This contrasts to overlaying abstract data metrics, such as speed or time, over a video, image or simulation, requiring viewers to consciously analyse. Instead, visual diffing utilises visual techniques and clues integrated into the scene which enable viewers to intuitively visually observe differences and areas of interest without needing to remember and compare abstract metrics.

Whilst not formally described as visual diffing in other domains, techniques which can be described as such can be found in sports coverage and a variety of other media, shown in figure 6.1.

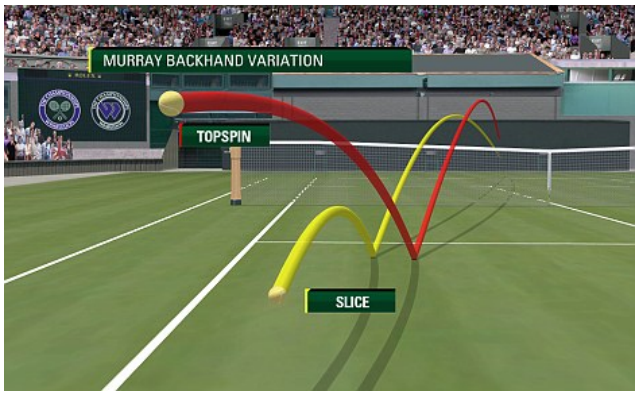
6.2.1 Techniques

The simplest of visual diffing techniques show spatial differences between entities in a scene, enabling observers to interpret the distance and difference in speed. Depending on the medium, image or video, different techniques are more affective for presenting information. Other techniques can help better translate non-visible data metrics, such as acceleration, speed, type, etc, to physical attributes, such as size and colour.

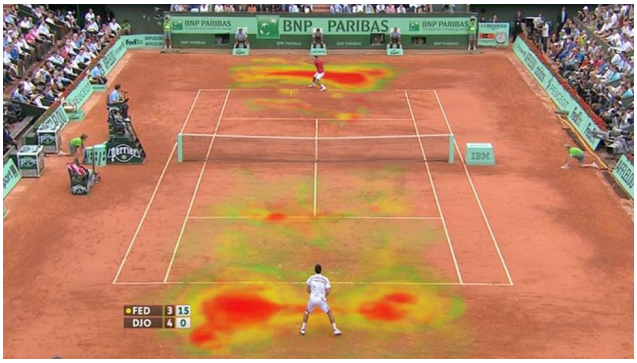
A non-exhaustive list of visual diffing techniques commonly used include:

[1] From which the term visual diffing is coined.

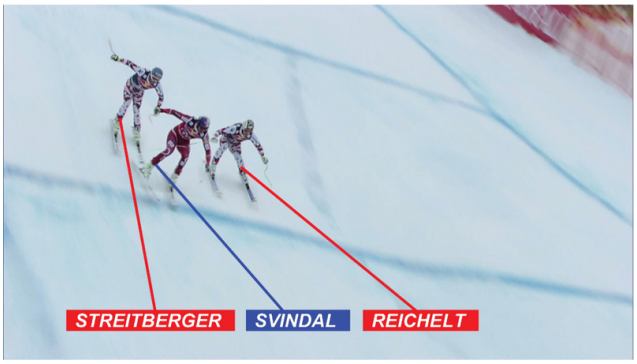
(a) The Hawk-Eye system showing where all service shots for a player land within the court.



(b) The Hawk-Eye system showing the path a shot took, to help determine if it is at fault.



(c) The Hawk-Eye system showing a heat map of player positions so far.



(d) Dartfish simulcam ski racing, showing three individual racer's time trials combined into one replay.

Figure 6.1: Examples of Visual Diffing within sports replays.

Ghosts[2], a common technique used in sports, in which two or more people or objects occupy the same on-screen environment, enabling visual comparisons of differences in position, speed and movement of participants. For example, shown in figure 6.1d; two individual slalom skiers racing in time-trials, are visually overlaid as ghosts onto the same race track. This provides viewers with a visually exciting but also intuitive experience as the race progresses, without the need to resort to just time metrics.

Paths, provide a visual history of where a person or object has been within an environment, typically represented as a line. Where ghost provides visual intuition at a discrete time-slice, paths provide it over a continuous time period. Within a slalom race, this enables viewers to observe what line a racer takes when skiing down a race course. These paths can be overlaid with others to compare racers directly, or even combined with ghosting to add more visually intuitive elements for the user to understand how a race progresses.

Colour + Size, enables non-visual data to be presented via a visual medium. Colour and

[2]The name ghosts comes from the ability of entities to pass through objects and each-other unaffected, not necessarily their ghastly colour, which is added to aid in disambiguation in DejaVu.

size can be used to display both discrete and continuous data, providing more information or highlighting hidden points of interest typically missed e.g., a set of colours can denote states, such as red, yellow and green, or a gradation of colour and size can be used to signify the strength or intensity of a data. Figure 6.1a utilise colour to provide meta-data about different tennis serves, whether they were first, second or ace serves.

Heatmaps, show the frequency of an entity or an action being carried out at a specific location within a space, through colour intensities displayed over a map. This provides at a glance where hot- and cold-spots of activity are within an environment, enabling viewers to instantly locate areas of interest. Figure 6.1c shows a heatmap overlayed on a tennis court, showing where the tennis players have returned shots since the start of the match, with higher frequency areas show increasingly intense colours, from green to yellow to red. Heatmaps are very effective for showing accumulative data

6.3 Implementing Visual Diffing

Implementing visual diffing techniques utilises the recording framework described in section 6.4.

6.3.1 Ghosts

The ghost visual diffing technique is the simplest technique visually, making it possible to view two simulation replays simultaneously in the same environment, with one simulation's entities rendered as solids whilst the second's are ghosts, shown in figure 6.2.

The simulator supports visual diffing a live simulation against a recorded replay, or one recorded replay against another. When visual diffing is enabled, the simulator creates "ghost" entities corresponding to the entities in the recorded repla,y which play out the recorded data from the replay. "Ghost" entities are not created for static objects in the environment, such as walls or floors - as these objects can't move.

In order for the "ghost" visual diffing technique to work, "ghost" entities must have a corresponding "live" entity (from the simulation or corresponding replay), so that the diff can be observed. "Ghost" and "live" entities are linked via id tokens. "Ghost" entities don't physically interact with the live simulation, simply replaying what was originally recorded. As the live simulation is run the the simulation time ticks at 60FPS; at each simulation tick the replay time updated and any snapshots that need to be applied are applied to the "ghost" entities.

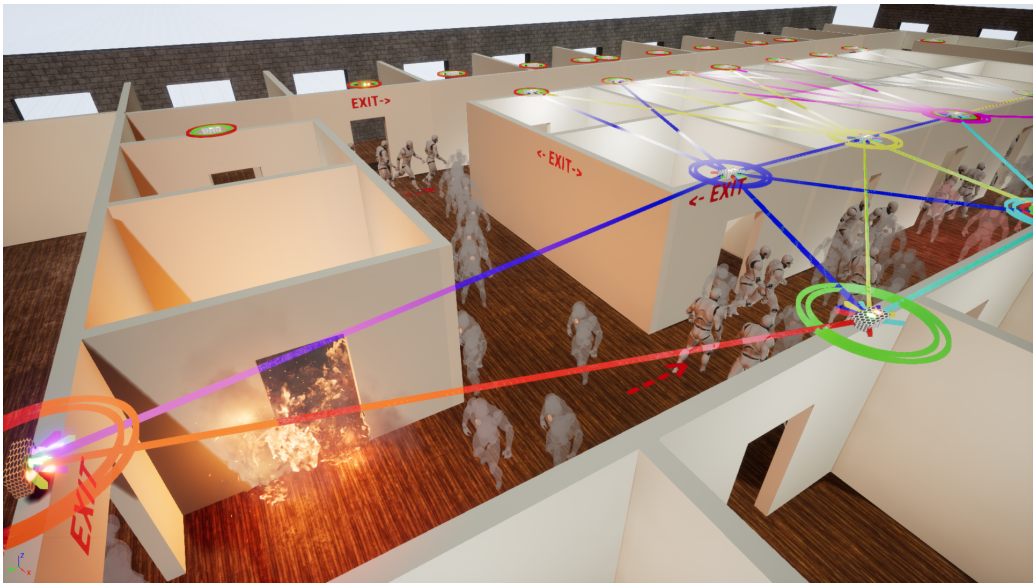


Figure 6.2: CPS-enhanced evacuation diffed with non-enhanced ghosts. Evacuees are being directed to exits via safe routes.

6.3.2 Paths

The path visual diffing technique displays the movement history of a entity as it traverses through the environment, leaving behind a trail consisting of lines and spheres, shown in figure 6.3.

Paths are generated in real-time from a recording, starting from time *t0*, a time sphere is created for each movable entity. New spheres are created at 1-second intervals, to reduce the granularity and provide visual time-step clues, joined by a line. Using this combination of spheres and lines provides a visual path of where an entity has been and the rate at which they were travelling. This makes it trivial to observe movement patterns and changes between different simulation runs.

For fast moving entities, it is possible decrease the interval time to increase the path granularity, displaying smoother paths through the environment.

The time spheres also provide a secondary time traversal function, time warping, described in section 6.3.3.

6.3.3 Time Warping

Time Spheres provide a secondary time traversal tool for visual diffing, time-warping. Within the virtual world times spheres are interactive, allowing observers to click on them to instantly time-warp between different time points.

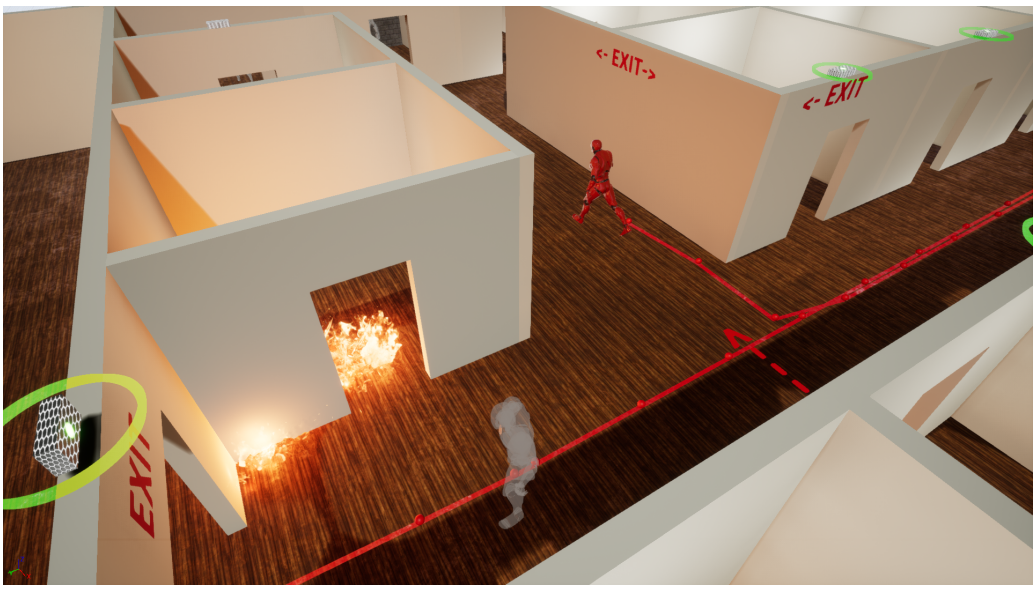


Figure 6.3: A person evacuating chooses two different paths in different runs based on CPS assistance, the ghost shows the same person taking a different route in a previous run.

As time spheres are generated they contain a reference to the time point at which they were created. Using this time point reference, the simulator can find and load the appropriate snapshot, resetting the environment to that snapshot's state. When this happens, the paths and time spheres remain, allowing for observers to time warp back and forth between time points.

6.3.4 Colour and Size

Ghosts and path visual diffing techniques support visualising spatial differences between entities, making it simple to spot differences over time or between simulations. However, visual diffing of internal state or non-visual data requires more creative visual techniques, such as colour and size.

6.4 Recording and Time Control

Running and recording CPS tests in real-world deployments is a difficult and time-consuming task, requiring not only recording and retrieving logs from a network of devices, but also video recording environments when physical phenomena are involved, such as people interacting with the system. Recording in the real-world is a "one-shot" chance, where anything missed, due to not being recorded in a log or within the video frame, can't be retrieved, requiring either guesstimating or a test re-run. Recording large volumes of data on-device may not be possible

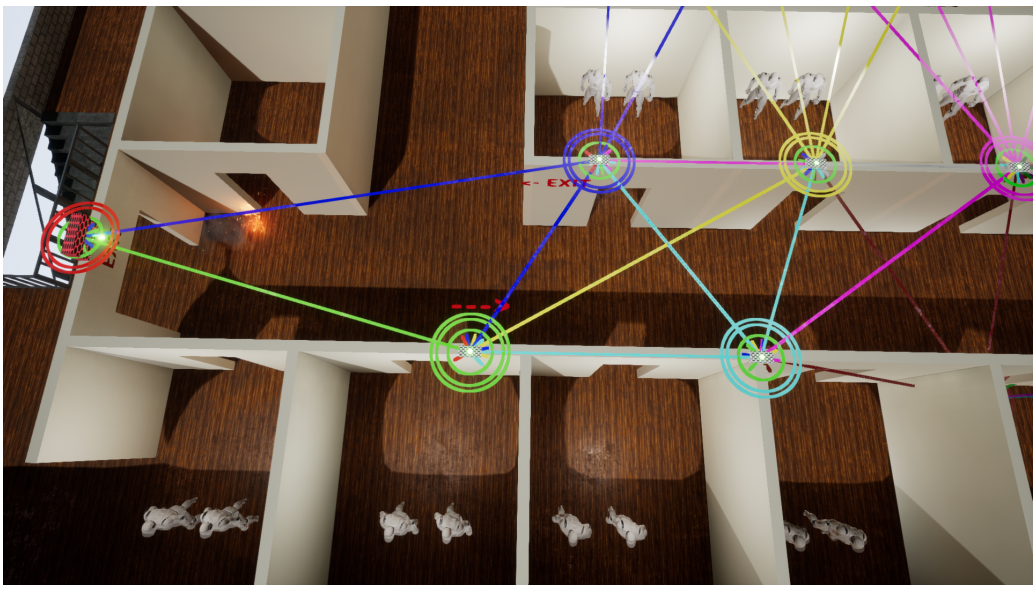


Figure 6.4: One of the sensors is red in colour and enlarged, representing a change in state when diffed to another run.

due to the constrained hardware, similarly, the network bandwidth may not support such data either, not without having adverse affects on the rest of the CPS.

Running simulations in a 3D virtual world opens up the opportunity for recording reconstructable replays, utilising data recorded from both the CPS simulation and from virtual world. Unlike other real-world or simulation recording approaches, these reconstructed replays allow viewers to see the cyber-physical cause and effect after-the-fact. Similarly, by recording at the simulation level, application code doesn't need to be edited to support it, ensuring application logic and control-flow consistency is guaranteed between testing and deployment.

Providing this level of replayability enables more comprehensive post-simulation review and analysis. For real-world deployments, simple video recordings which may be carried out and are limited to the recorded view(s) and are read-only once recorded. Instead, by recording reconstructions, viewers are able to manipulate time and space within replays, pausing, fast-forwarding and replaying events from any angle within the scene. Reconstructions also allow for analytics to be seamless added to or removed from the replay in real-time, such as ghosts, providing a dynamic and flexible tool-kit e.g., changing viewing angles/position and altering what visual diffs are applied.

Recording long-term simulations provides different challenges, not only requiring efficient storage of the reconstruction, but also how to efficiently access, index and review the points-of-interest. Solving these issues opens up the opportunity to run longitudinal CPS studies for testing robustness and reliability, even against running systems e.g., a real running system could run parallel to a upgraded simulated version, feeding inputs and testing whether the new version maintains the same functionality.

- Gigabytes of data per hour, how to store, compress, index...
- Synchronising recordings, replay, rewind, fast-forward efficiently.

6.4.1 Recording Data

Simulation within the co-simulation platform targets 60FPS to provide a smooth and responsive simulation. Thus, recording enough data to support creating reconstructions matching the original run requires logging the attributes of every entity within the world and CPS simulator at 60x per second, including location, rotation, speed and other entity-specific internal state. This can generate GBs of data per hour without optimisation, limiting simulation recordings to several hours on an average workstation.

The following sections describe several optimisations and techniques used to improve the space-saving yield whilst maintaining the smooth performance.

Raw

In order to record full reconstructions, the simulator needs to record both visible and non-visible behaviour i.e., both physical and internal state. Such data includes, position, rotation, trajectory, sensor data, transmission state, etc. By ensuring as much information as possible is stored, future reviews are able to view and process not just what developers thought might have been necessary when they ran the test. However, this extra information comes at a cost, as shown in figure 6.1.

At each time slice, a snapshot of an entity's state is stored as part of its history, in chronological order starting from time *t0*. Thus, in order to rewind to a previous state, the simulator simply sets the entity's state equal to the snapshot. For moving to a specific time point, the simulator uses the time stamp as an index to locate the correct snapshot, then simply sets the state to that.

Revisit how it moves to a specific time point

Delta Optimisation and Checkpoints

To reduce the amount of data recorded, instead of storing a snapshot of an entity at every tick (60FPS), we can instead store one only when some or all of its state has changed, thus saving considerable amount of space for entities which become inactive, physically or virtually; however, for simulations where all entities are active, the savings may be limited.

Table 6.1: Recording costs of objects within the simulator

Data	Size	1min Total	1hr Total
Location	16Bytes	56KBytes	3.2MBytes
Rotation	16Bytes	56KBytes	3.2MBytes
Scale	16Bytes	56KBytes	3.2MBytes
Velocity	16Bytes	56KBytes	3.2MBytes
Angular Velocity	16Bytes	56KBytes	3.2MBytes
Time Stamp	4Bytes	1.4KBytes	84KBytes
Sensor State	39Bytes	137KBytes	8MBytes

When using this technique, searching for specific time points becomes more difficult as there may not be a corresponding snapshot for an entity, instead the last recorded snapshot before the requested time point should be used.

To further improve upon this, each snapshot only contains what has changed between time slices, starting from time *t0*. When rewinding or fast-forwarding through a continuous time period, these delta snapshots can be applied like before, except updating only the changed state.

However, providing random access to the history becomes more difficult, due to the snapshot containing only what has changed, if any snapshot exists at all for that time point. To jump to time *tn*, the simulator must start from *t0* and apply each delta snapshot up to *tn*, resulting in an O(n) time. Alternatively, at a set interval – checkpoint, a full snapshot could be stored, enabling faster state restoring, requiring finding the nearest checkpoint to time *tn*, before then applying each update until reaching *tn*. This reduces the time to O(n/m), where m is the number of checkpoints, thus, a higher number of checkpoints reduces the search time, but will increase the storage consumption.

TODO ▶ Run experiment to see difference between optimisation and compression

Table 6.2: Size comparison between raw and compressed snapshots saved to disk, varying in length.

	1min	1hr	24hr	7days
raw	40MB	1.5GB	36GB	252GB
optimised	**TODO**	**TODO**	**TODO**	**TODO**
compressed	1.6MB	45.1MB	1.1GB	8GB

6.4.2 Enabling long-term recordings

With the current recording solution, even with the delta optimisation, it's possible to run out of memory within a few hours, if a given scene has many entities which are all very active,

moving and changing state regularly. Therefore, to record, replay and analyse long-term CPS simulations consisting of days or weeks of content further improvements are needed to provide efficient loading, viewing and storing of these recordings.

The rest of this section discusses several techniques employed to improve performance and usability when using long-term recordings.

Snapshot Chunking, Compression and Time-Shifting

Currently, when viewing a replay sequentially, the replay file can be streamed into memory, providing O(1) access to replay content. However, the current format doesn't support random access, requiring the whole replay to be loaded into memory up to the time point of interest, O(n) time, where n is the time point. This is not an optimal solution given replays can be in the order of hours to days.

When replaying a long replay the system needs to sequentially load in the recording until the desired time-point is reached. This makes viewing small segments of hour/day/week long simulations inefficient.

To solve this, snapshot chunking is used. The continuous stream of snapshots both full and deltas, are divided into chunks or segments of a fixed size. Segments which are not currently being viewed are compressed and written to storage (disk or ssd). When running a replay the initial segment is loaded and the rest of the segments are set to lazy load in on-demand. This technique significantly reduces the memory requirements for recording and viewing long term recordings.

Compression is used to significantly reduce the size of segments whilst on-disk. As segments are saved to or loaded from disk they are compressed or uncompressed, respectively.

Because of the optimisations used for storing snapshots, equal length time slices can contain significantly different numbers of snapshots, which would result in vastly different snapshot sizes e.g,. fixed 30sec snapshots: snapshot 1 contains 100 snapshots, whereas due to inactivity snapshot 2 and 3 contain only 20 snapshots each. This causes unpredictable loading performance and memory usage when accessing and de-compressing segments, with smaller segments loading quickly versus larger ones taking significantly longer and consuming considerably more space. Alternatively, storing fixed size segments ensures predictable memory usage and loading speed.

Pre-emptive segment loading is used to improve sequential and short-distance time-shifting performance i.e., segments before and after the current segment are loaded. This allows the user to jump forwards or backwards crossing into adjacent segments without causing significant slow

downs caused by waiting for segments to be loaded in from disk. The number of pre-emptive segments loaded can be adjusted based on the available memory and performance requirements.

Utilising these techniques makes long-term recordings feasible, significantly reducing the space and time required to store, load and view recordings, from over a gigabyte for 1 hour to under 50 megabytes.

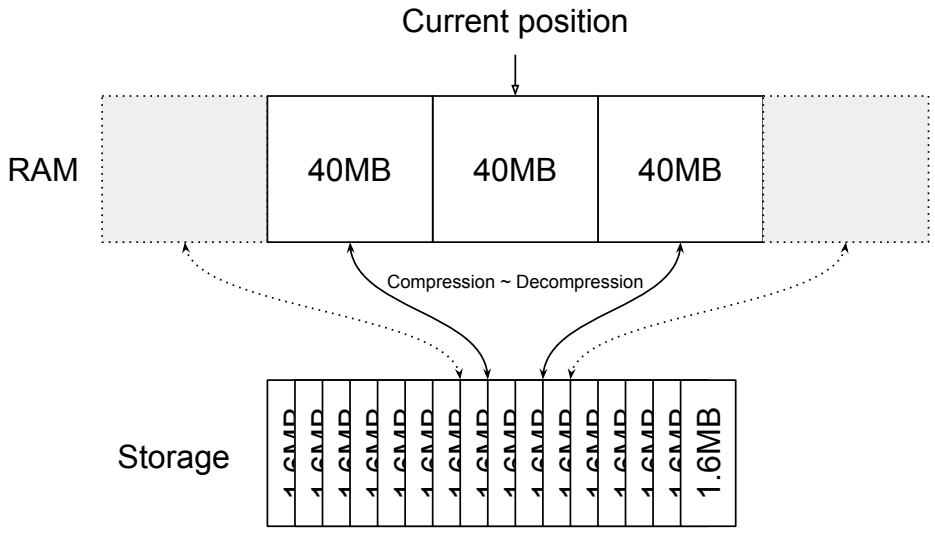


Figure 6.5: Segmented compression scheme employed by DejaVu to reduce in-memory usage.

6.4.3 Finding Points-of-Interest

Recording, storing and viewing long replays are made possible by the chunked compression recording scheme described above, however, generating this much replay content requires reviewing it too; when reviewing there are often periods of interest when certain activity or no activity may be occurring, but in order to find those interesting points users must watch the footage until they find that point in time, a time-consuming task which defeats the benefit of simulation being faster than real-time. Therefore, some form of automatic indexing of points-of-interest is necessary, enabling developers to instantaneously time-shift there without the need to search.

6.4.4 Built-in PoI

To resolve this issue, we developed a set of built-in listener functions which trigger point-of-interest checkpoints in the timeline when the event of interest occurs. Using this checkpoint developers can time-shift directly to it to see and understand what happened. Using these

functions developers can instruct the simulator to watch for certain attributes or events to occur e.g., watch for when node 1 receives a message, when a person is moving/stops moving.

These functions are set to run once per tick to access the necessary simulation data and determine if a condition has been met. These functions include checking: when a node transmits, when a node moves, when a nodes LED state changes, when a PIR sensor has been triggered.

Ideally developers should be able to write functions which can access the event stream and then generate custom PoI checkpoints.

Pub/Sub PoI

Because the framework is built upon a Pub/Sub message bus, it makes it possible for external tools to subscribe to the simulation event stream, perform stream processing and pattern match for events of interest. Using this developers can write custom event watchers without recompiling the simulator code.

To make this work the simulation components publish world and simulation events to the message bus, to which external components can subscribe. The simulator also subscribes to the PoI stream; when a developer's event listener finds a match it can then publish the timestamp and event name to the PoI stream; when received by the simulation component, it records this PoI checkpoint for later playback.

Utilising this Pub/Sub architectural approach decouples developers from the simulator codebase and language, enabling developers to write custom scripts/tools in larger number of languages and tools.

6.5 Full Time Control

6.6 Case Study

To demonstrate the visual diffing tools we developed a case study, deploying a cyber-physical-enhanced fire detection and evacuation system. This CPS caters for large buildings with multiple corridors and exit routes, such as offices or shopping malls, in which finding the safest route out is non-trivial.

Previous work by [6, 5] has demonstrated the use and benefit of utilising a 2D evacuation simulator to develop new cyber-physical systems to enhance detection and evacuation, ensur-

ing people are directed away from high-risk areas, reducing congestion at critical paths and providing detailed information for emergency crew to locate areas of high-interest.

Our case study utilises the algorithm from this approach, to provide a walk-through of, demonstrate and evaluate the visual diffing techniques and time control features, such as time-warping and checkpoints.

```
broadcast msg {ID, dist}
select:
  waiton msg {ID, dist}:
    routes[ID] = dist + 1
    effectiveDist = min(dist+1, effectiveDist)
    if changed:
      broadcast msg {ID, effectiveDist}
  waiton msg {ID, FIRE}:
    routes[ID] = FIRE
    effectiveDist = min_hazard(routes)
    broadcast msg {ID, effectiveDist}
  waiton sensor {FIRE}:
    broadcast msg {ID, FIRE}
```

Figure 6.6: Pseudo-code for distributed evacuation CPS

The algorithm, shown in pseudo-code form in figure 6.6, runs on each node within the network. Each node first broadcasts its distance to an exit; upon receiving a distance message, a node recalculates its shortest path to an exit; upon receiving a fire message, a node removes the senders path and recalculates the shortest distance to an exit taking into consideration the distance to the closest exit via a path and that path's hazard risk based on its distance to the nearest hazard; lastly, if a node receives a fire message from its own sensor, it broadcasts the fire alert to the rest of the network. Each node knows the direction of its nearest neighbours at deployment time, thus, using light indicators, observers can be directed along the calculated safe routes.

Using DejaVu, we interactively programmed the evacuation system, testing and debugging our initial distributed navigation algorithm using virtual agents to react to virtual emergency fire evacuations. Using the replay and diff features, we were able to observe the differences that occur between different runs of the evacuation. For our analysis we focused on observing evacuation navigation between simulation runs, i.e., agent movement in reaction to the algorithm direction. Further examples of DejaVu's diffing capabilities are show in the appendix.

Upon adapting the fire evacuation algorithm, we are able to use DejaVu to visually observe how it affects evacuation navigation. Before using the algorithm, evacuees would simply navigate to the exit closest to them, signposted by static exit signs, as shown in figure **??**, unless they encounter a fire. When the CPS is deployed, shown in figure 6.2, evacuees are guided to the nearest safe exit, which may be longer than the shortest route, because of a fire hazard en

route. Using DejaVu, the difference between these two scenarios is clear, in which the unaided scenario can be seen played out by the ghosts, whilst the current CPS-enhanced scenario avoids directing people towards the fire, using dynamic floor signs - pointing the people towards the safe route.

Sensors can also be seen as white cuboids mounted along the corridor walls in figure 6.2, detecting fire and relaying information between themselves according to the algorithm. The double coloured circles and lines signify radio traffic from one node to another, whilst the smaller coloured rings and LEDs reflect the status LEDS the node; these visual overlays expose more information typically hidden from sight in the real-world.

6.7 Evaluation

- How does it perform?
- Is it possible to compare 2 or more in real-time or even faster?

6.7.1 Compression

6.7.2 Real-time Performance

6.8 Conclusion

Chapter 7

Analysing a CPS in the real world using AR (15 Pages)

7.1 Introduction

7.2 Challenge

Why is it challenging?

- How can we transfer what we learned in the previous chapter to the real-world?
- Can we use AR to perform real-time analysis of a real/virtual CPS
- How to communicate between AR and CPS in real-time, without significantly impacting CPS?
- Can we create a transferrable solution?

7.3 A window into a CPS using AR

- High volume of data, how to communicate back and forth?
- Synchronising visual display, locating CPS.

7.4 Tracking a CPS in the real world

7.5 Design

7.6 Case Study Analysis

7.7 Evaluation

7.7.1 Real-time tracking and analysis

7.7.2 Impact on CPS

7.8 Conclusion

Chapter 8

Conclusions and Future Work

8.1 Summary

8.2 Similarities, Limitations and Trade-offs

8.3 Summary of Thesis Achievements

Summary.

8.4 Applications

Applications.

8.5 Future Work

Future Work.

Bibliography

[1] Blender 3D Game Engine and Modelling Program. `http://www.blender.org/`. 2015.

[2] CryEngine 3D Game Engine. `http://cryengine.com/`. 2015.

[3] Unity 3D Game Engine. `http://unity3d.com/`. 2015.

[4] UnrealEngine 3D Game Engine. `http://www.unrealengine.com/`. 2015.

[5] O. J. Akinwande, H. Bi, and E. Gelenbe. Managing crowds in hazards with dynamic grouping. *IEEE Access*, 3:1060–1070, 2015.

[6] Nikolaos Dimakis, Avgoustinos Filippoupolitis, and Erol Gelenbe. Distributed building evacuation simulator for smart emergency management. *The Computer Journal*, 53(9):1384, 2010.

[7] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 126–137, New York, NY, USA, 2003. ACM.

[8] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648, Nov 2006.