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 use of wireless simulators for calculation location placement based on environment [29]. Discuss the use of copying radio RSS from a real network[20] 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[2], Unity 3D[9], Unreal Engine 4[10] and CryEngine[5], 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 a developer's capability and coverage of testing.

This chapter presents a CPS co-simulation framework designed to unearth tools from their silos to create a flexible, extensible and powerful co-simulation platform to close the cyber-physical loop in CPS development, simulation, testing and analysis. 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.

3.1 The Ideal Framework: Primary Requirements

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 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 if it reacts as expected. However, exercising control over the real-world can be a time-consuming and sometimes impossible challenge. Similarly, enlisting human participants to perform repetitive, mundane tasks requires paperwork, significant time and often money. Some scenarios are simply not feasible to test due to their scale, limited access to a deployment environment or health and safety concerns, e.g., peak office hours, outdoor environments, evacuation, fire, flooding.

The framework should provide a realistic real-time physics engine to create dynamic, controllable and flexible phenomena on demand, which interact with a simulated or real cyber-physical system. Phenomena could be created through either direct control, in which a developer directly interacts with the simulation by taking control of an entity or agent; or scripted control, in which a developer instructs agent or the environment to carry out or perform certain actions at points in time. For example a developer could take control of an agent and walk them down a corridor manually, instruct a group of agents to navigate along a path at set intervals, directly pick up and move a node, or trigger a fire to start and spread along a set of rooms via a common corridor with agents reacting accordingly. All of these phenomena will dynamically generate physics events that can trigger sensors within a CPS; the CPS can then perform actuation in the virtual world, closing the CPS loop.

A scripted phenomena, such as a moving object, mobile node or person walking down a corridor, dynamically generates sensor input in real-time as it moves. Thus, only the phenomena script

needs to be updated to change an entity's path or behaviour. On the other hand, when using traces or scripted input, a developer must update multiple traces to reach the same effect: feeding sensors simulated "detections" inputs, for each sensor device in the entity's path, a time-consuming and unscalable solution.

3.1.2 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 is static and pre-determined; in other words, developers define mobility patterns, as a sequence of positions, manually, which the network of devices must follow, but these devices do not react to the live cyber-physical simulation. The tools have no concept of other entities, such as people, physical objects or gravity.

The tools lack of support for simulating human mobility, such that developers are required to manually trigger sensors or create custom scripts which attempt to model the mobility for each test scenario, in which the script triggers button presses or motion sensors at specified 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 and reactive 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 in real-time for the CPS to sense and react to, without the need to edit scripts or manually interact with the system, even as it grows, shrinks or changes in layout. Participants are also reactive to changes caused by the CPS, such as closing a door, forcing the participants to navigate around obstacles or change their behaviour.

Using such a system, when simulating a fire evacuation scenario, it would be possible to create

behaviours in which 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. As the fire is started in different locations, the CPS and participants react differently, avoiding areas of danger, which were previously safe.

Exploring human behaviour further, in various scenarios the concept of physiological and cognitive attributes can also have a significant impact when simulating human mobility behaviour[28], such as age, fitness, assertiveness, risk-aversion, panic and fear[30]. Age and fitness of an agent can dictate their speed and vulnerability during an escape, causing them to slow down if there are old, unfit, injured or have a disability. Cognitive attributes, such as assertiveness or aggressiveness may cause them to charge through a path, forcing others out the way, or they may form or influence a herd. Alternatively, risk-aversion, panic and fear may cause agents to ignore evacuation signs, stay put or egress via the entrance they entered the building, even when passing sign-posted exits that offer a more rapid evacuation[28, 27, 37]. Relationships can also have an affect on the evacuation speed of occupants: as parents wait or search for children, slowing down their escape and blocking others; or a group of friends or colleagues attempt to form before evacuating an area, again slowing themselves and other down.

3.1.3 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 achieve 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. Thus, virtually experimenting with placement ahead of time may help mitigate these issues before deployment, testing out new layouts or configurations.

Testing different placement strategies within a 3D virtual world would enable developers to quickly move nodes around an environment programmatically or by visually dropping them into place, without the need to manually change scripts and sensor input traces; instead, the virtual

world dynamically generates sensor inputs in real-time for nodes based on their new positions. Similarly, developers should be able to flexibly scale the levels of device provision, to test and understand how it will affect a deployment.

Building upon this, the system should support automatically testing and suggestion of optimum placement and provisioning levels, based on selected criterion, such as cost, quality-of-service, robustness, coverage, etc.

To improve accuracy of location dependent sensors, such as temperature, light, or radio performance, this approach could be combined with in-the-field gathered trace data, from either deployed devices or an augmented-reality tracked mobile node.

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, including the state of the virtual world, the agents and devices within it, 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, e.g., when a network partition occurs.

On top of recording the states of visible entities, such as agents or devices, it's vital to record events that occur with in the CPS, such as a radio transmission, motion sensor trigger or device actuation events. The resulting event stream can then be used to provide additional context to

the virtual world replay, used as trace data for other simulation runs, or replayed via other tools in the framework.

3.1.5 Visualisation - Visual Diffing and Analytics

To complement log output, existing CPS simulators provide basic 2D visualisations, such as 2-dimensional network maps and transmission timelines[35, 6], giving developers more insight into the network component in real-time. However, visualisation development has only focussed on visualising networking information, with other information only presented in the form of logs or tables. Visualisations in existing tools are ephemeral and can only be viewed once as the simulation is on-going, anything missed is lost and the simulation must be restarted to view it again. Visualisations can be recorded, however, once recorded into video format the visualisations can no longer be explored, tweaked and investigated for more in-depth information that they usually contain, e.g., clicking on a node in the network map reveals more information about the node.

Furthermore, if a developer wants to compare two separate simulation runs, they must resort to post-simulation log analysis, typically using external tools to manually process the log output into another form, visual or otherwise, without the context that the simulator visualisation tools provide.

Firstly, visualisations should provide visual context to developers both during and after a simulation, with the capability to explore and investigate previously unseen data or snapshots, thus, simulators or the visualisation tools must record generated visualisation data in a replayable format.

Secondly, visualisations should not just help developers understand a single simulation but should also help developers to spot and analyse differences, “visual diffing”, between multiple simulation runs in real-time, without the need the need to leave the tool to perform post-simulation data analysis using manual tools. When simulating in a 3D virtual world, visual diffing will empower developers to see, explore and analyse cyber-physical systems in the context of the virtual world and past simulation runs.

Visual diffing tools should 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.

Lastly, 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 layered on top of the 3-dimensional virtual world, akin to a lap or split time on a racing video, 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.

3.1.6 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 sense 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.

A device's sensors should be modelled using the framework's physics engine to provide dynamic and realistic sensor feedback, reacting to phenomena in the 3D virtual environment in real-time. Similarly, actuators should enable devices to physically interact with the virtual environment to close the cyber-physical loop in simulation e.g., opening a door, turning on a light, or sounding an alarm. Alternatively, it is not necessary for devices to match or be constrained by their real-world counter-parts, thus, sensors may be modelled more accurately or powerful, enabling them to sense faster, with greater range or more fidelity; or less accurately depending on the testing scenario needs and the simulation performance available. One could go so far as testing out futuristic sensors that may not be available on the market, conceptualising different approaches given a new type or variation of sensor, e.g., indoor identifiable personnel tracking.

Visually, device hardware can be modelled at varying degrees of accuracy, depending on the

requirements, i.e., virtual devices can be larger in size compared to real counter-parts, enabling easier location and observation. Alternatively, if device motion and movement is important, the devices can be modelled to accurately reflect the real-world device's size, weight, shape and surface texture, each of which can affect the physics of how it may move and interact with the environment (momentum, friction, collisions).

3.2 Secondary Requirements

3.2.1 Environment-based Radio Simulation

Ideally the framework should allow for the export of the built 3D environment used for virtual deployments, to provide environment-based radio simulations tools the same 3D environment model for generating a realistic radio model. Using the exported 3D model, the radio simulator could generate a realistic radio model based on the physical environment, such as walls, objects and other structures, and its attributes, such a material type, thickness, orientation, reflectivity.[29]

3.2.2 Distributed component-based simulation

Rather than building one monolithic tool, the co-simulator should be discretised into components with clear interfaces; such that the architecture allows for distributing components across multiple machines, to enable better performance and flexibility, such as running visual aspects on a mobile device whilst the simulation runs on a workstation PC. Using the approach, the architecture would naturally have support swapping out virtual components for either others virtual components, or their real counterparts e.g., swapping out a set of virtual simulated nodes for a set of real deployed hardware nodes; this would provides developers the platoform to seamlessly test real vs. simulated nodes and potentially use visually diffing tools on them to further analyse their differences.

3.2.3 Extensibility

The framework's distributed design should facilitate seamless integration of new tools to supplement or replace the existing ones, through the use of open networking protocols and a well defined common schema. This will enable future developers to build upon and improve the co-simulation framework, its output and experience through the use of interchanging or adding additional tools, including simulation tools, analytical tools, model checkers or incorporating real devices in test-beds or deployed in the environment.

3.2.4 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 WSN simulation methods which aren't 100% accurate and the intricacies of OS level scheduling, synchronisation is necessary to ensure that the individual component's clocks stay in sync with one another, within reasonable bounds. Dependent on the level of testing, synchronisation bounds may be more or less strict, in which case 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.

3.2.5 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.

3.2.6 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.

3.2.7 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.

3.2.8 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.

3.2.9 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.

Alternatively, using augmented reality technologies, such as a Google Tango-enabled device[26, 31] (Lenovo Phab 2 Pro), it would be possible to use the Simultaneous Localisation and Mapping (SLAM) abilities built-in to build a virtual version of the environment simply as you walk around. This virtual map would not only be centimetre accurate but the colours and designs (textures) of the walls and objects in the environment would match what the camera recorded. This would significantly reduce the complexity associated with designing accurate virtual environments, instead simply requiring a single pass through a space in order to virtually recreate it.

3.2.10 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 a range of environment and network sizes, from tens, hundreds and even thousands of devices and from a single room, to a building, up to city-scale spaces. 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.

Chapter 4

Framework

The current state-of-the-art tools for cyber-physical systems focus on individual aspects of testing, such as network or hardware simulation, but are siloed from other tools. Many other tools and platforms exist that could support these CPS tools with other aspects of CPS testing, such as environment and physics simulation, formal modelling, or statistical analysis tools, without the need to directly extend or integrate these features within any of them.

Hence, the goal of this framework is to provide an open, stable and extensible platform into which a variety of tools can be integrated to support the simulation, testing, modelling and analysis of cyber-physical systems.

This section proposes an architecture for designing a distributed framework supporting a co-simulation approach for testing and analysis of cyber-physical systems using multiple tools. This will enhance the information sharing between tools, enabling developers to seamlessly use these tools in tandem 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.

4.1 Architecture and Communication

Rather than integrate different co-simulation components together directly, our approach uses a publish/subscribe event bus and common schema to describe information passed between the different tools. This approach reduces the tight coupling between tools, allowing for individual

tools to be removed, added or replaced with minimal configuration. We envision the integration of tools such as model checking, statistical analysis, unit-testing and advanced simulation for radio and environmental properties.

Each tool publishes or subscribes to the relevant topics of interest within the event stream, enabling other tools to observe or inform one another of events in a many-to-many fashion. Typically, only one tool will publish data to a topic related to that which it specialises in, e.g., radio simulator publishes when transmissions were sent and received. Other tools subscribed to this stream may then publish a composite event, adding more data, analysis or context back to the event stream.

To support the platform each tool needs to build a plugin which communicates between the event bus and tool itself, subscribing to and publishing data. A tool's plugin defines a set of topics to which it publish/subscribe to, providing developers with a clear interface between co-operating tools.

4.2 Time

Within the simulation world there are two clocks, the simulated time and the real world or wall clock time, how much time has passed in the simulation reality and how much time has passed the the real world. Comparing these two times gives us the simulation rate, i.e., how much simulated time passes in 1 second of real world time. It's often the case that simulations can run faster than real-time, resulting in a simulation rate >1x, providing results faster than had they been carried out in the real world. Similarly, users can also select to run simulations slower than real-time, giving developers more time to observe and analyse the simulation in real-time.

When performing co-simulation between different components or tools, it's necessary to ensure the simulation tools remain synchronised, such that the individual simulations aren't adversely affected, e.g., the CPS simulation lags behind the virtual world simulation, resulting in delayed and slow CPS response times, causing devices in the virtual world to actuate incorrectly.

Also, should a developer choose to run a simulation at a speed faster or slower than real-time, it is also necessary to ensure the co-simulation tools support this functionality and are instructed to simulate at the desired speed.

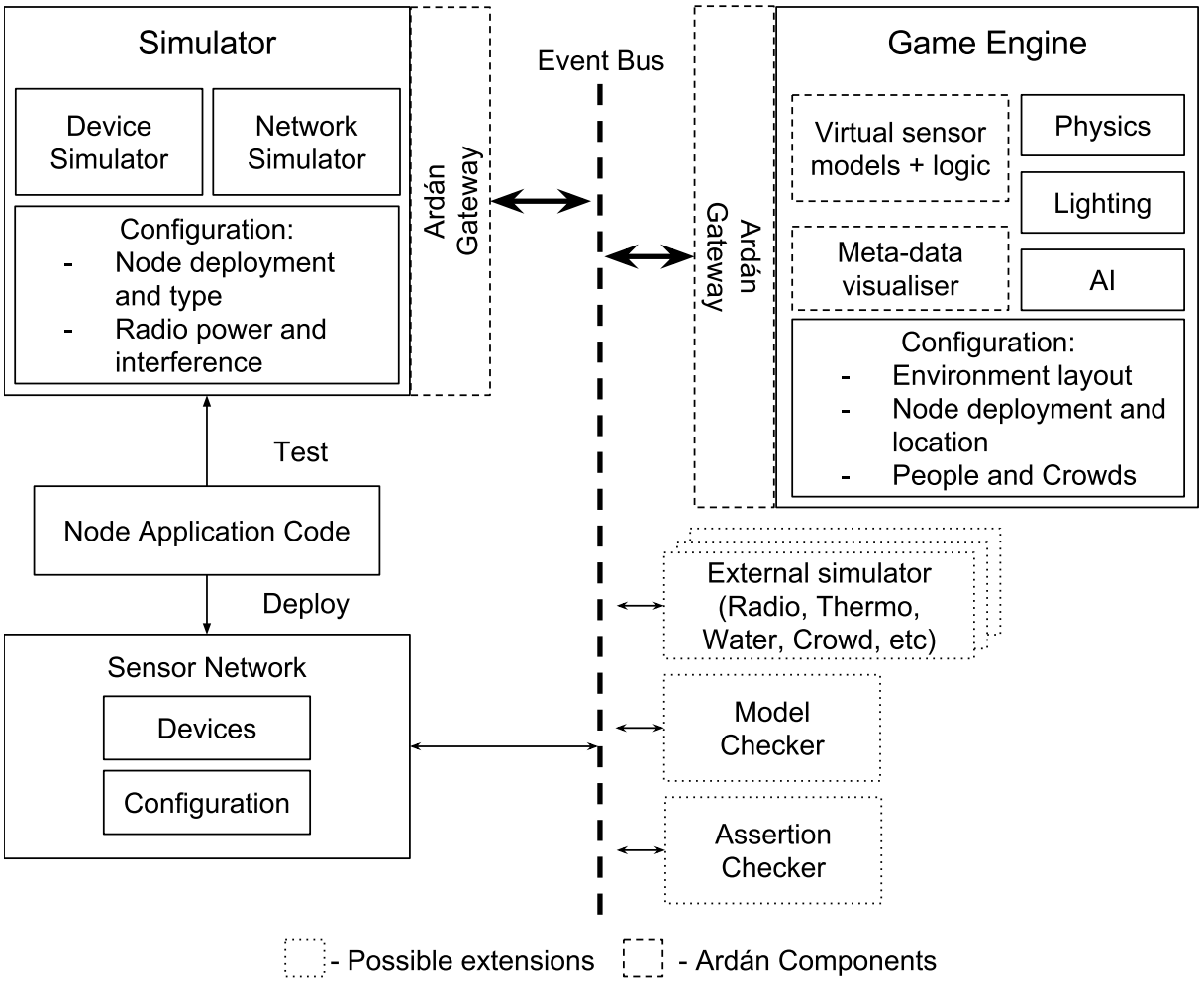


Figure 4.1: Ardán architecture

4.2.1 Human Mobility

Within Ardan, the game engine component is responsible for human mobility, handling the physical modelling and characteristics, such as size, weight, speed and appearance, as well as their behaviours.

When creating a virtual person, henceforth referred to as a agent, developers can choose to create or spawn them in one of three ways: programmatically in C++, spawning an agent specifying their appearance, scale (size) and location; interactively, using the visual game engine editor to select a agent type, drag and then drop into the environment, before then assigning further attributes; scripting in the game engine's visual programming tool, similar to C++ in its API and programming logic, however, it provides developers with a visual drag-and-drop based programming tool, with useful auto-suggestions and completion.

Utilising the physics engine within the game, an agent's properties, such as size and weight, can have a significant impact on their interactivity with the world, such as their body fitting through spaces or their weight impacting their momentum and force in collisions with other objects. To navigate the physical environment, agents use a navigation mesh built by the physics engine, which calculates traversable paths for agents to use to navigate within the environment, around obstacles, through doorways and up ramps or stairs. Agents then use the A* algorithm on the navigation mesh to find the shortest path to their destination.

After spawning an agent, behaviours can be assigned to them which they will perform at run-time. Behaviours, similar to spawning an agent, can be created and customised using the previously mentioned methods. The visual programming tool simplifies this process, by enabling developers to interactively find and select entities of interest and create sequences of conditions and actions to form behaviours. Developers have access to a large API of primitives to support complex behaviours, such as sphere and ray tracing, used to detect and measure nearby and visible agents or entities. Behaviours can be as simple as move to a location, requiring an agent to navigate the environment and avoid any obstacles; or more complex, such as agent X follow agent Y, requiring the agent X to determine where Y is before than navigating to their position.

Agents

It is also possible to create crowd behaviours, such as herding, in which a group of agents move and perform actions as a collective unit, without any central decision making. These behaviours can be constructed by creating agent behaviours that are influenced by other agents and their behaviours. Within the game engine a herd may start due to an agent following someone who they deem has authority in a situation (manager, teacher). The herd may also influence other agents to join in due size of the group, 'flocking'. For example, within a fire evacuation a manager may form a herd with colleagues leading them to a safe exit, other agents observing would choose to follow even if the route was neither the safest nor quickest.

Group behaviour, family/friends waiting/searching

Occupant aggression, physiological differences

Chapter 5

Case Study

5.1 Introduction

To aid this thesis, two case studies are presented below, describing the use of an indoor CPS to improve two services for users of the building; the first service targets reducing indoor lighting energy consumption, whilst the second provides dynamic navigation for emergency evacuation.

5.2 Energy Efficient Corridor Lighting

Lighting left on in our schools, universities and offices wastes not only millions of pounds a year but increases our carbon footprint, with an average[1]office lit overnight consuming that equal to the energy needed to boil over a 1,000 cups of tea[8, 4].

To tackle this, the use of smart / intelligent lighting can decrease the amount of energy wasted during off peak times, such as evenings and weekends, when few people occupy them. Similarly, we can also decrease wastage during summer periods, for offices in which bright sunlight is available. Socially, using automatic lighting prevents issues regarding lighting control over large office spaces, in which there may only be a single main control, where social pressure may prevent a single person from turning off unneeded lights over a shared-space.

Deploying and testing a scheme such as this is a costly and risky challenge, with budget concerns, safety regulations and any on-peak downtime or faults can be costly to businesses or public

buildings. On top of this many different technical concerns which would need to be addressed, including, where to place sensors, algorithm robustness/performance, lighting comfort, to name a few. Thus, thorough testing is absolutely required, but, testing in the real-world is limited by the aforementioned concerns.

5.2.1 Specification

The goal of the CPS lighting is to significantly reduce the out-of-hours lighting energy consumption while ensuring the remaining lighting left on is comfortable for any occupants still occupying the building. This can be achieved in two ways, for areas in which there are:

No occupants, turn off lights completely in the area. For example, an empty office space or floor. As occupants move towards an unoccupied area, lights immediately around the detected occupant are gradually fully lit as they approach, whilst, more distant areas in line of sight are gradually lit. This ensures occupants are able to see around the area they occupy without the shock or distraction of lights flickering on and off.

Some occupants, ensure lights around occupants are set to full/desired brightness, but reduce lighting intensity in the surrounding area. Exit routes, such as corridors, have reduced lighting but remain visibly lit, ensuring occupants feel aware of their surroundings and comfortable.

To ensure the office spaces remain comfortable to work in, the lighting scheme must perform lighting changes in gradual steps: individual lights are always gradually turned on or off, never from completely off to completely on, or vice-versa, limiting the occupants to sudden and distracting changes in light; the light levels of neighbouring groups of lights nearby occupants gradually reduce relative to the distance from the occupants, reducing the energy consumption whilst ensuring occupants feel comfortable.

5.2.2 Environment

This case study is based on a typical closed-plan office space, with an array of small rooms connected by several corridors. Rooms are shared by 4 occupants. Lights are evenly distributed along corridors and within rooms.

[1]Based on a typical office space of 100m2 with 18 x 6ft (1800mm) T8 tubes at 70W each.

Test scenarios focus on the period of off-peak office usage in which minimal lighting is used, to reduce energy usage. Lighting can be assumed to be using dimmable LED lighting in which the colour and intensity can be adjusted instantaneously, unlike traditional florescent tubes which can't be dimmed and typically have a 2-3 second delay before turning on (usually flickering).

5.2.3 People

Office-based participants typically occupy their personal office spaces for prolonged periods of time (in the order of hours), occasionally traversing between rooms for meetings, lunch, walks or toilet breaks (in the order of tens of minutes).

5.2.4 Sensors + Actuators

In order for the CPS to balance energy reduction goals and comfort levels, sensing technology must be used to detect the presence of people; two main sensor types could be considered, people sensors and light sensors. People sensors can be used to detect occupancy of spaces and indicate areas in which lighting can be reduced. Light sensors can be used to detect ambient light levels, to adjust lighting levels accordingly. This case study will focus on using only people sensors to alter light levels.

Within the real-world we are limited by the technology available today and the limitations individual sensing technologies provide. Sensing presence within an indoor environment can be divided into three main categories.

Person identification - Enables a CPS to identify and recognise individual participants, with varying levels of spatial accuracy e.g., cameras can detect position based within their field of view, RFID tags are effective at detecting presence in areas in which entry/exit requires scanning a tag, WiFi/Bluetooth provide identification based on a user device within range of the sensor.

Presence detection - Enables a CPS to detect the presence of people within an area, typically a wide-range PIR sensor, via the use of thermal detection. Typically not able to differentiate between 1 or more people. Able to capture presence in relatively non-active environments, e.g., people sitting at desks typing.

Movement detection - Can detect simply the movement of people within a space, typically detects walking and running.

Depending on the location type, corridor or office, different sensors may be more appropriate. Utilising a presence or person sensor within an office may work effectively where movement may not be as large or consistent, in which people may be stationary or seated for pro-longed periods of time. On the other-hand, in a corridor, movement detection (motion sensors) would be accurate enough to detect passerbys.

Within a 3D game engine the sensing method can be abstract, not limited by a specific technology or sensor type, enabling the CPS to detect the presence of people at varying levels of granularity, i.e., person identification, presence detection, movement detection.

5.2.5 Placement

A key concern when designing a system like this is ensuring consistent and reliable coverage of both occupant detection and the wireless network. If the occupancy detection fails in an area, due to a blind-spot or unreliable detection, this can lead to a poor user experience, leaving occupants in the dark. However, deploying too many sensors will result in redundant data and increased costs. Hence, testing placement within the target environment is absolutely necessary to mitigate the associated risks.

Sensors will be deployed in each room and along the corridor. Various sensors with different ranges can be tested, short range, long range, etc. Depending the on the sensor range, more or less sensors can be used, with differing placements to ensure complete coverage.

5.2.6 Privacy Concerns

The design of this lighting scheme proposes monitoring of the occupants' location within a building, knowledge of which is usually already available to building managers, typically via existing security infrastructure, such as CCTV or personal RFID entry cards. However, in buildings with outward facing windows and smaller, more personal offices, it may be possible to observe and determine the location of specific individuals externally. This case study does not focus on this concern.

5.3 Fire Evacuation using Distributed Navigation

Fire evacuation design and planning is a key part of all public buildings, including schools, offices and hospitals, in which ensuring there is a prescribed fire evacuation plan in place, with safe and clearly marked routes to urgently egress a building in the event of a fire or other emergency.

As part of building regulations [7], architects and building managers must ensure there are adequate escape routes proportional to the size and capacity of a building. Escape routes must be clearly sign posted around the building, providing directions to the closest exit at any given point. These signs are typically printed or illuminated signs, providing only a single fixed direction towards an exit.

However, there are a variety of issues with the current signage approach. In the event of an emergency research has shown people often ignore emergency exit signs and instead attempt to egress via the entrance they ingressed the building or via a familiar route [37]. This is often not the fastest route, causing evacuees to travel a further distance to egress the building. Additionally, this behaviour has shown to cause significant congestion en-route and at the exit caused by a significant majority of people acting similarly. The herding effect can amplify this, with people ignoring logic and signs, instead following the herd[37, 27], which may be leading people in the wrong direction. In the case of a fire en-route to the nearest signposted exit, it may not be possible to determine the safest route to without additional knowledge.

This case study focuses on demonstrating a CPS-enhanced improvement to these signs; the concept involves deploying a CPS-enhanced signage, which dynamically calculates safe routes to aide evacuees in navigating along safe evacuation routes during a fire alert. Safe routes are displayed along the floor and walls of the corridors.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.

5.3.1 Layout

Nodes will be placed along corridors, doorways and/or rooms equipped with fire detectors (smoke, heat, etc). Nodes will also be equipped 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.

5.3.2 Sensors + Actuators

In order for the CPS to help evacuees navigate safer routes towards an exit, the CPS needs to be able to sense where a source of danger is, such as fire. To further enhance the evacuation algorithm, the CPS could consider the congestion level along pathways and exits, preferring to direct evacuees along less congested, safe, but longer routes, relieving congestion at the problem area.

5.3.3 Specification

The goal of the fire evacuation CPS is to dynamically calculate the shortest safe route to an exit, displaying this to an evacuee via a direction sign.

Some basic invariants we want to ensure for such a system is:

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

The distributed dynamic evacuation navigation algorithm is not novel to this thesis, previously described by Gelenbe et al.[14]. The CPS will be deployed as fire sensors throughout the corridors of a virtual office space, defined in section 5.3.1. When a sensor detects fire, it broadcasts this to the network. Nodes which receive the message, blacklist the node's associated direction if it lies on a path to an exit, the message is forwarded on and subsequant nodes perform the same blacklisting action. Upon blacklisting, the node recalculates its shortest safe route, taking into consideration the blacklisted nodes and their proximity to the route.

```
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);
  }
}
```

Figure 5.1: Evacuation navigation psuedocode algorithm

Chapter 6

Simulating Cyber-Physical Systems in the Virtual-World

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 or creating the desired phenomena to interact with the network. 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. Human mobility also poses difficulties, including participant recruitment, health and safety requirements and limitations, physical limitations, etc. Hence, performing tests in the real-world is time-consuming, costly and often impractical.

In this chapter we present a novel co-simulation approach to this problem, designed taking into consideration the the requirements discussed in chapter **??**. The co-simulator is built upon a publish-subscribe event-bus, integrating a high-performance 3D game engine, Unreal Engine 4, with an existing sensor network simulation platform, Cooja, to create a dynamic, flexible and more reliable end-to-end simulation solution for testing cyber-physical systems in their target environments. Harnessing a 3D game engine, we introduce simulated cyber-physical systems into virtual reality, with realistic real-time physics, dynamic and controllable phenomena, AI agents, and realistic lighting effects. The co-simulation platform adopts the publish-subscribe architecture, supporting an open, flexible and expandable platform for co-simulation development.

6.1 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. Significant research within WSN and CPS simulation has focussed on accurately simulating and emulating hardware devices, the network and power consumption, with great success [34, 32]. Alternatively, test-beds offer testing with real devices and over a real radio network, sometimes with tunable network inference[36] or other conditions, such as temperature[17]. However, these tools and techniques used to develop, test and analyse such systems don't simulate the physical environment, its inhabitants or their mobility.

The rest of this section discusses the issues related to simulation and test-beds in more detail.

6.1.1 Simulation Issues

When testing in simulation, typically the environment data is fed into simulations using recorded sensor data (traces) from an existing deployment, field study[11, 23], or is fabricated based on a developer's conceptual model and testing needs. However, using this approach is not without issues.

Acquisition of traces is a difficult and time consuming task, as traces need to be recorded from existing real-world deployments or field studies; the former may not match the target network, and the later can be time-consuming, expensive, dangerous, or inconvenient to perform thoroughly. Once these traces are recorded, they are static and dependent on the fixed position they were recorded from, unable to be modified after-the-fact, e.g., to adjust for moving a sensor in a simulation.

In the case of fabricating traces, these are highly dependent on the desired phenomena a developer is attempting to model and their understanding of it, hence, modelling complex phenomena such as gravity or thermal dynamics by hand is a non-trivial task. Instead developers may choose to imitate certain phenomena with less precision or accuracy, which may result in a disconnect between the test and its performance in the real-world.

Regardless of the method for creating traces, creating comprehensive traces which cover a large number of test scenarios across different scales and layouts of a network is extremely time-

consuming and often not feasible, e.g., gathering real traces on a bridge with limited access [23]. Hence, the total coverage of tests for these different variables are reduced.

6.1.2 Deployment and Test-bed Issues

For CPS involving human and device mobility, deployment and test-beds[13, 18, 17, 36] are not only time-consuming and expensive to deploy, but also are limited by health and safety, people recruitment, location availability, and scalability issues.

Testing involving people needs to ensure strict health and safety guidelines and laws are adhered to, requiring participants to be safeguarded against possible injury and death, thus, testing situations in which poses a threat to participants can't be carried out, e.g., testing crowd based scenarios or scenarios involving hazards such as fire or flooding. Similarly, ethical guidelines must also be followed when involving people, requiring developers to consider guidelines such as VIP ethical guidelines used in human computer interaction (HCI) studies[16, 15]: Vulnerability, care must be taken when recruiting participants young, elderly, or people with particular conditions; Informed consent, participants must be aware of the experiment and be allowed to withdraw at any time; Privacy and confidentiality must be observed, with participants data anonymised.

It's not possible to run tests with real people in real evacuation scenarios, due to the health and safety risks mentioned above. Instead we can try to simulate these scenarios in the real-world using fire-drills, a significantly lower risk to participants; however, fire drills by nature have an impact on how people may act during an evacuation. Similarly, people may become familiar with the environment the more tests they perform, which could affect their reaction time and the choices they make.

Recruiting participant also poses a challenge for larger test scenarios, in which recruiting, organising and managing large numbers of participants requires significant planning, approval and management.Similarly, running large numbers of tests with even a small number of people may simply not be possible due to time constraints, thus, only a core set of tests may be carried out instead, limiting the scope.

Exclusive access to target locations prior to full deployment for testing can be difficult at peak times, requiring out-of-hours testing on weekends or evenings. This can have a knock on impact for recruiting participants.

In the rest of this chapter, we introduce a novel CPS co-simulation platform that has been developed to tackle the issue of simulating a CPS with realistic, dynamic and flexible environment input, described above; the design of the co-simulation platform is based on the framework described in chapter **??**.

6.2 A Virtual Reality Co-Simulation Platform

To meet the needs of simulating a CPS pre-deployment with realistic, dynamic and flexible environment input a novel approach is needed, capable of simulating both the cyber, a CPS, and the physical, the environment, humans and phenomena; hence, we designed and built a co-simulation platform, integrating a WSN simulator with a 3D game engine to close the cyber-physical loop in simulations.

6.2.1 The 3D Game Engine: Unreal Engine 4

Using the 3D game engine, we're able to create vivid and realistic models of the real world, involving complex physics, AI pedestrians and crowds, physical destruction, and lighting. In games, films and TV these effects are used purely for creating immersive experiences which can trick the viewer into suspending disbelief; within a simulation these can be instead used effectively to create living simulations grounded in reality.

Using the game engine's physics engine, we're able to build 3-dimensional physical environments, from the size of a corridor up to a whole city, complete with realistic physical properties and effects, such as gravity, momentum, friction and collisions between objects in real-time. Leveraging the real-time physics we can create dynamic and reactive physical input for our cyber simulations. Using the physics engine we can also simulate destructible environments and buildings, dynamically reacting to virtual earthquakes, explosions or other destructive phenomena.

Like many game engines, the Unreal Engine also includes a suite of tools to support creating responsive AI characters, capable of dynamic and reactive behaviour in real-time. Utilising these tools we can simulate human-centric CPS such as in offices, schools or stadiums, with pedestrians and crowds exhibiting desired behaviours, such as goal-based movement, avoidance, herding, panic, etc.

Unreal Engine can also simulate realistic lighting effects, such as refraction, bloom, shadows, fire and smoke; lighting can have a significant impact within an environment, such as a room or building, providing participants with a clear view when natural or artificial light is available or poor visibility when fire and smoke fill a room. when performing human-centric simulations, allowing viewers to understand how lighting and visibility may be affected by different sources of light (sun, lamps, lights, fire) and in different scenarios, day-to-day office, fire evacuation etc. Using these features, we can:

Using the input from the 3D game engine to drive the CPS simulation which can then drive change in the virtual world, creates truly dynamic and reactive closed-loop simulations; such a system removes the need for developers to gather, fabricate, modify or update individual sensor traces. This gives developers more time to focus on building better CPS based on their tests, than building the tests.

We believe utilising a 3D game engine to drive the CPS simulation can help resolve the simulation issues described previously, section 6.1.1. Developers can model 3D environments which realistically match the design, layout, scale and conditions of their desired target environment, from single rooms to whole buildings. This enables developers to capture sensor traces from anywhere in the environment and virtually experiment with device placement at any time, without the difficulties associated with accessing real deployment environments.

Leveraging the 3D game engine's physics engine, it's possible to create dynamic, reactive and realistic sensor traces, relieving the burden on the developer to create "realistic" sensor traces. For example, calculating when a person walking at 2km/h will intercept motion sensors lining a CPS-enhanced lit corridor. Because of this testing out various scenarios is scalable as the number of devices or scenarios increases, as devices or AI are added, moved or removed, the simulation will dynamically adapt and create new traces for the relevant sensors.

Using the 3D game world it's possible to run tests not only at any time, no longer restricted by real-world access hours, but also possible to reduce testing time by simulating the virtual-world at faster than real-time.

Similarly, we believe this co-simulation approach can begin to address the deployment and testbed issues described in section 6.1.2.

Virtual tests involve virtual people, hence, health and safety rules need not apply. However, one

envisions they could apply to some extent if real people are used for virtual reality (VR) tests, but with much less risk attached e.g., real people can't be physically burned by a virtual fire, although they may experience VR sickness.

Participant recruitment is a non-issue, within a 3D game engine world, 100's of AI participants can be created as and when needed, no bribing or ethics paperwork necessary.

As discussed above location access is resolved by modelling it in 3D, based off a floor-plan, requiring no access to the real physical location. Alternatively, a one-time measurement survey can be carried out with minimal interference caused.

It is possible to run tests with large numbers of people 24/7, without issue or complaints, scheduling issues or location availability issues.

Whilst achieving 100% realistic people and crowd behaviour may not be possible using a game engine's AI, we can use AI people and crowds to mimic previously observed behaviour accurately, enabling more consistent simulations when compared to real people in fake or "drill" scenarios, e.g., virtual people can be told to panic or mimic herding behaviour, whereas real people may not exhibit this behaviour in non-life-threatening scenarios. This enables developers to test for these different behaviours and scenarios which affect them. However, the AI behaviour may not reflect the true unique and complex reactions of a real person and crowd in real scenarios.

6.2.2 The WSN/CPS Simulator: Cooja

Like previous physical simulator approaches [14, 33], we could use a standard programming language and environment to program and model the WSN component, such as the language used within the game engine, C++. However, this approach has several drawbacks.

Programming in a non-WSN or CPS language and environment requires porting to the desired platform when transferring from testing to deployment. This can introduce translation bugs and vary in difficulty depending on the differences in language, environment and especially on the differences in programming paradigm or memory model e.g., event-based or imperative, static or dynamic memory.

Similarly, many issues in WSN and CPS depend on the hardware and performance constraints of the deployment devices, which have severely limited CPU, RAM, ROM and battery power.

Hence, simulated nodes without consideration or an accurate simulation of these constraints, i.e., running natively on a full desktop machine, won't provide an accurate representation of how the software will run once deployed on a set of constrained devices.

WSN/CPS rely on their radio network for communicating between nodes within a network, which also has a significant impact on other aspects of the device, such as performance, duty cycle and battery life.

Utilising the Cooja WSN simulator enables the co-simulation platform to build upon the tool to create a powerful testing platform which developers can test on before seamlessly deploying to real devices in their target environment.

Cooja nodes are programmed in the Contiki C language, hence, the same code can be deployed for simulated nodes as for deployed nodes. This enables a seamless transition between simulated and real nodes and removes the translation bugs and issues related to translating between a simulated language and a real node language.

Cooja nodes can be simulated at various accuracy levels, using the same code, dependent on the simulation level needed (hardware emulation, simulation or natively) and the available computing power of the host simulation PC. Dependent on a developers needs this can help test accurately using hardware emulation, or test scalably using less hardware-accurate simulation but with significantly larger network sizes, 10's vs 100's devices.

Cooja simulates the radio medium with several optional radio simulation models, enabling developers to test simple and complex radio scenarios, with varying RSSI and interference models to reflect real-world radio environments.

6.3 Co-Simulator Design

Discussed in chapter **??**, the co-simulation platform is designed in an open, component agnostic fashion, i.e., the platform encourages the interoperability of different simulation platforms, specifying clear abstractions and boundaries between components and uses a publish / subscribe bus to communicate. The goal is to create a plug-n-play platform that can support the use of different simulation, analytical and game engine tools, with our implementation providing an example construction.

6.3.1 WSN Simulator Component

The WSN simulator, Cooja, performs the hardware, software and radio network simulation for each node within a WSN in the co-simulation. Users select the hardware they wish to deploy there application code, software, to, which is then compiled for the desired platform. Depending on the level of accuracy, host PC performance and size of the desired network, developers can select the hardware simulation level they wish the node to be simulated at, which ultimately dictates whether the code is compiled to the target or host architecture. This enables developers to sacrifice hardware accuracy for a larger simulated network.

Radio

Radio simulation is performed entirely within Cooja, relying on the built-in radio models and distance between nodes to determine node RSSI and packet-loss. However, to enable dynamic and mobile nodes, Cooja subscribes to node location updates; when a nodes position is updated by other components, such as by being moved in the virtual world, the radio model is updated any any subsequent radio activity is affected by this.

To provide other components with information about the radio environment, including transmissions and successful or interfered receives, the radio component publishes all radio state to the radio topic.

Utilising the game engine's knowledge of the 3D world and obstructions between nodes could improve the fidelity of the radio simulation, however, this remains as future work. Kokkinis et al.[29] have demonstrated the use of 3D models of target environments to simulate an accurate radio model based on obstructions and their material types. Once the 3D model is built within the game engine component, the model can be exported to the TruNet tool to calculate the radio model and then import back into Cooja providing an accurate radio model based on the virtual environment.

Location and Movement

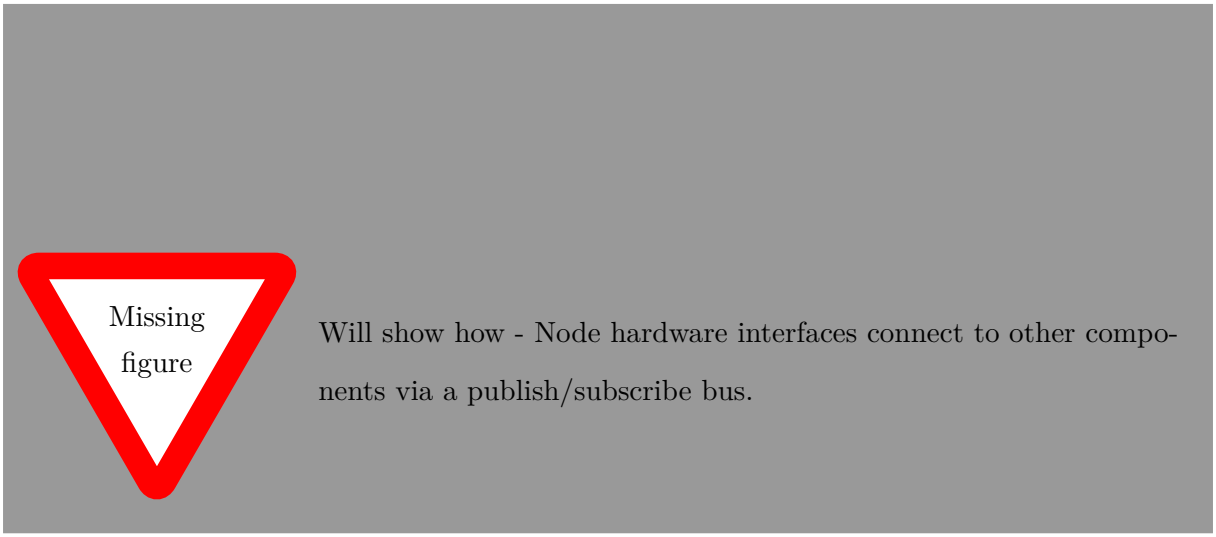
Each node is configured with an initial 3D location, used in the radio simulation to calculate radio interference. The location is exposed to other co-simulation components via pub/sub topics

which Cooja is subscribed to. This enables it to be updated at any time by other co-simulation components, in this case the game engine. This provides developers the ability to test scenarios with dynamic mobile nodes, which affect the radio simulation within Cooja.

It is also possible for nodes with inertial measurement units (IMU) to receive updates, to inform applications of any 3D movement, discussed further in the following section.

Hardware Interfaces - Sensors and Actuators

Hardware interfaces which normally interact with the physical world, such as motion, acceleration, light sensors and buttons, are exposed as virtual hardware interfaces to other co-simulation components via pub/sub topics.



Data related to input interfaces can be published to by co-simulation components, to which Cooja subscribes and distributes to the relevant nodes, such that when a node queries an on-board sensor, the most up-to-date reading is available, from within the component. This also ensures response time for hardware interface requests is kept to the absolute minimal, requiring no cross-component calls (i.e., network calls) to retrieve data. However, this method could result in applications reading delayed, stale data, by approximately the RTT/2 between publisher and subscriber; in applications where sensor data is high frequency and ephemeral, this approach induce a time-delay effect when comparing simulated vs real-world tests. This delay will be explored in the evaluation, section **??**.

Similarly, data related to output interfaces, such as LEDs or actuators, is published by Cooja to the relevant per-interface topics; allowing subscribed components to be updated in real-time

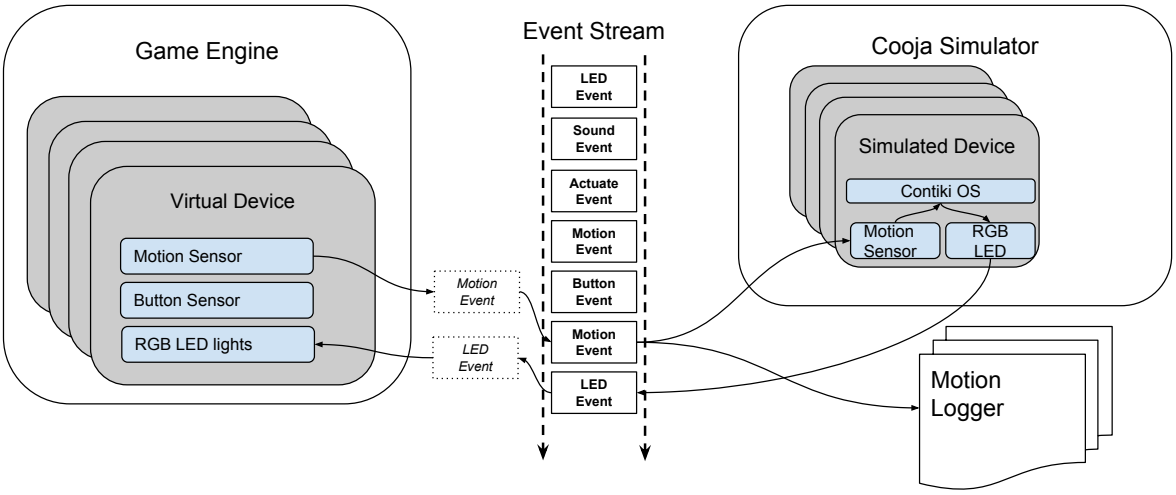


Figure 6.1: Node Event bus

of such hardware events, which can then be: actioned in a virtual world component, such as turning on a light, opening a door or activating an alarm; or recorded by a logging or analytical component for further analysis.

6.3.2 3D Game Engine Component

The 3D game engine component, Unreal Engine 4, is responsible for simulating the 3D virtual world and the physics, lighting and people within it. It also models the physical manifestation of the nodes and their exposed hardware interfaces i.e., sensors and actuators.

Virtual Node Model

Within the virtual world we can choose to model node hardware to an accurate scale, or not. This allows for developers to choose physical accuracy when required for scenarios which may involve mobile nodes that can be affected by physics and collisions with the environment. On the other hand, we can sacrifice realism by increasing a nodes scale; this can help the visual locatability of a node and visibility of any information on the device, such as LEDs or an ID.

Figure 6.2 shows three different scales of the basic sensor node model we designed. The model is designed as a simple cuboid with large LED lights on the top of the device, matching those typically found on a sensor node (TelosB Mote).

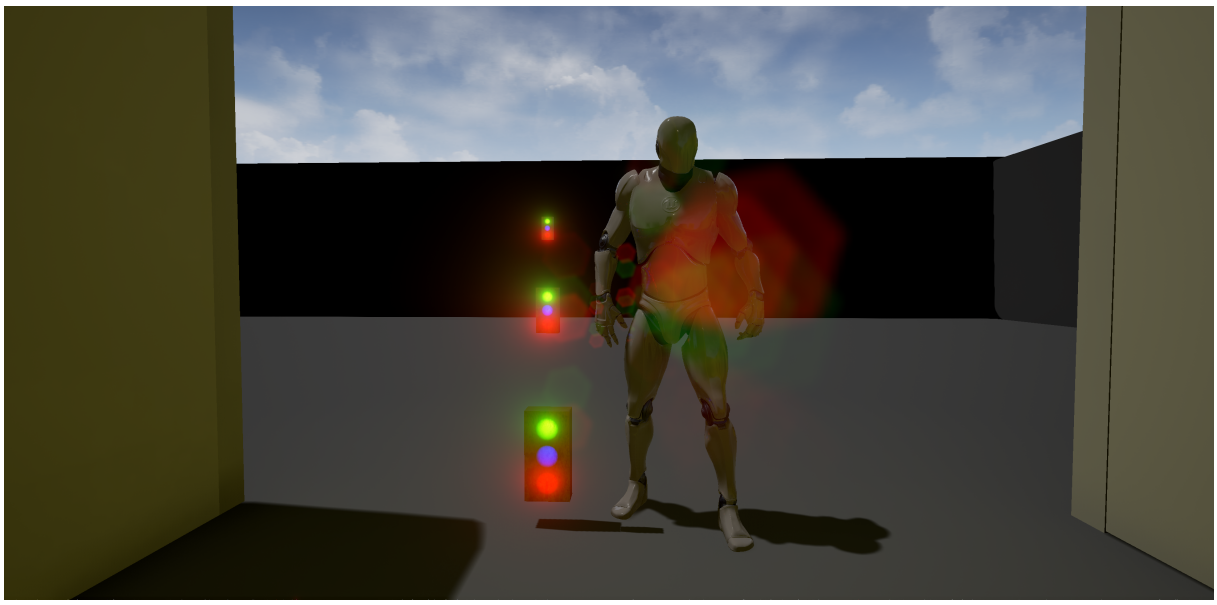


Figure 6.2: Variable sensor scales, providing either physically accurate sized devices or more visible devices

Within the virtual world, nodes can be set to either: static, its location is fixed and is not affected by physics; moveable, it can be acted upon by the physics engine e.g., fall due to gravity, react from a collision; or attached to another entity, such as a person or vehicle, moving as they move. This enables nodes to be dynamic mobile nodes reacting to the virtual environment.

Virtual Node Hardware Interface

A node's representation within the game engine can sense and actuate within the virtual world via their virtual hardware interfaces. A node can be assigned multiple interfaces, each of which can publish or subscribe to updates on the relevant pub/sub topics with their associated node's ID.

In the virtual world these hardware interfaces can programmatically sense the virtual world using primitives available within the game engine, shown in figure 6.3, such as trigger boxes, detects when entities pass in and out of an invisible non-colliding 3D shape; collisions, triggers an event when an entity collides with something in the environment; and ray casts, casts a line from a point in space, typically perpendicular to the face of an object, detecting any objects which it intersects with along the line's path.

Figure 6.3 demonstrates these different types of primitives available to developers to simulate physical sensors. Starting from the left, an agent is seen walking down a corridor and passes

into a cuboid shaped trigger box, which upon happening, triggers an event, much like a motion sensor. Within the game engine the trigger box can be any shape, is invisible, allows objects to pass through them and each event provides any objects associated with causing the trigger - figure 6.4 shows a conical shaped trigger box simulating the field-of-view of a typical motion sensor. Using a trigger box, one could simulate a simple motion sensor, detecting when objects pass through its field of view, or enhance it by enabling it to recognise the names or types of objects or people which pass through it. Moving to the right, a ball is falling and colliding with another cube, this time triggering an event as it collides and ricochets off, upon colliding it's possible to detect the force of impact and any objects associated with the impact, providing the ability to simulate a sudden motion or vibration sensor (piezo sensor). Lastly, on the right is a ray-cast, starting from an object and passing through the sphere and person. Ray-casts have a configurable distance, stopping at either the nth object or at a set distance, returning all objects it intersects, from which a distance can be calculated.

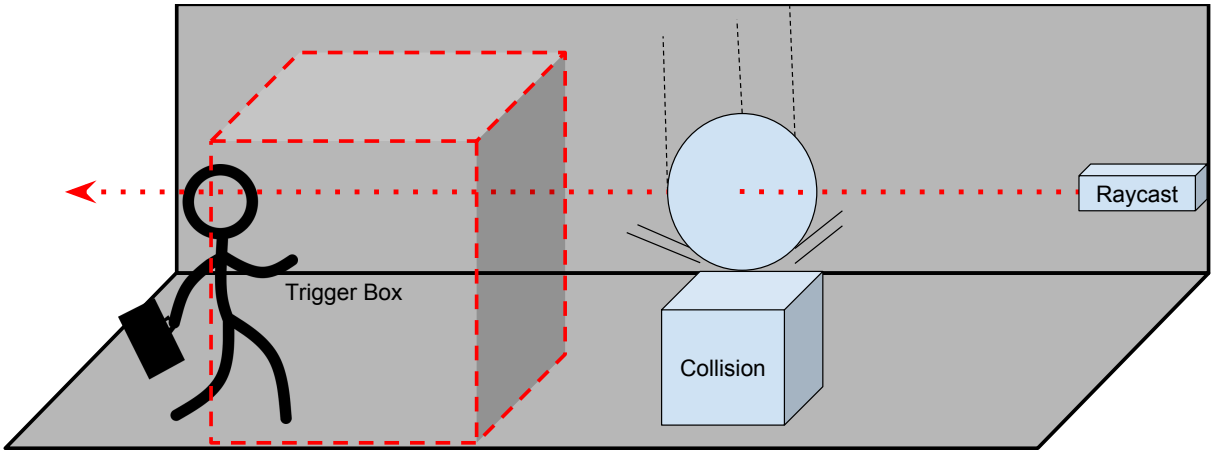


Figure 6.3: A visual example of three different types of physics detection primitives available in a game engine: trigger boxes, collisions and ray-casts.

Using these game engine primitives, developers can model a variety of sensors to desired varying degrees of accuracy e.g., a PIR sensor could use a trigger box to detect when a person enters its range, a distance sensor could use a ray-cast to detect distance to the first object it intersects, or a vibration sensor could use collisions to detect vibrations forces when it is hit. These and other primitives give developers the power to experiment with different sensor types, accuracies and capabilities, which may not be available to them or exist. Sensors are assigned to a virtual node's sensing port, associating a sensor with a unique node ID.

For performing actuation, a virtual actuation object, such as a light, alarm or door, is assigned

to a node's actuation port. By default a node exposes 3 standard actuation ports, one for each LED built into a typical node. To perform actuation, another component simply publishes to the relevant actuator topic, which the targeted node then receives and performs the actuation in the virtual world.

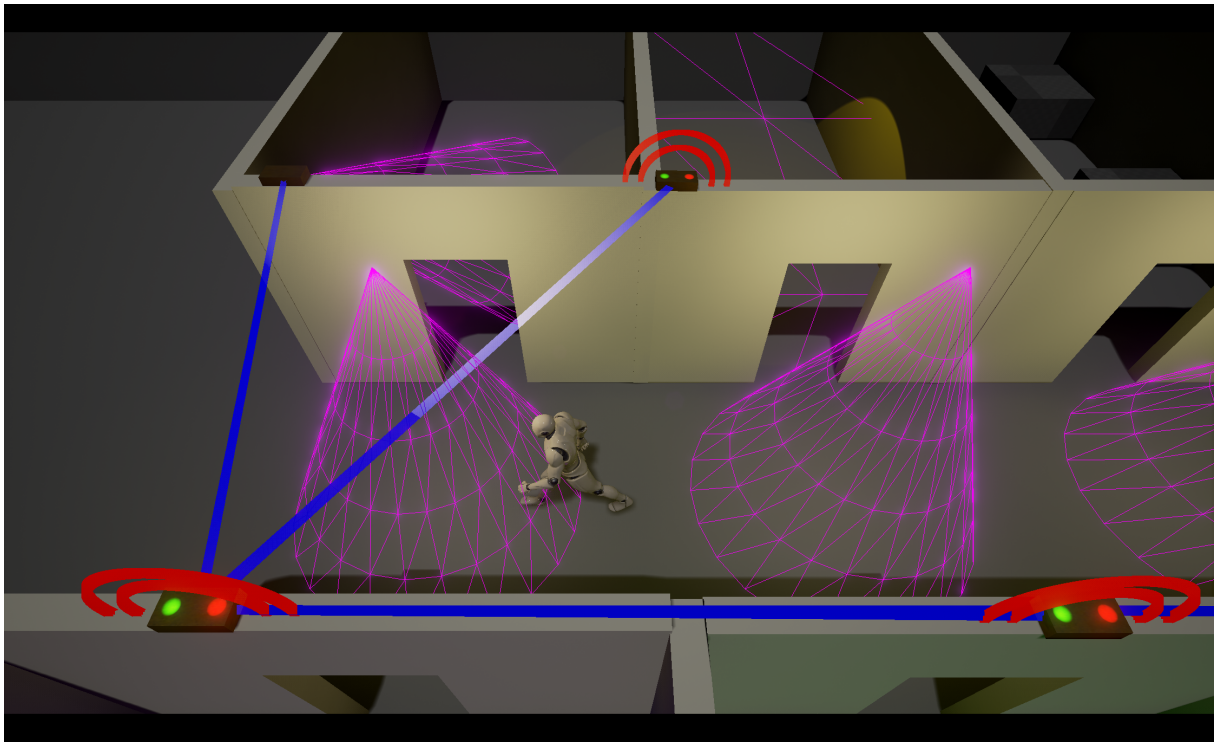


Figure 6.4: Motion sensors with their field-of-view visible.

Within the co-simulation platform we consider the following initial sensor and actuator types:

Button - A physical button sensor can be directly interacted with by people, both AI and controlled, when in range. When pressed an event is triggered in the game engine.

Type: Triggered by an event.

Distance - A distance sensor uses a single ray-cast to measure the straight-line distance from a point until the first object it hits. The distance sensor runs at a specified frequency, e.g., 20Hz.

Trip - A trip sensor uses trigger box, several pixels wide and the length of the space, which upon an object crossing into it triggers an event within the game engine.

Motion - Similar to a trip sensor, a motion sensor uses a trigger box in the shape of the motion sensor field-of-view, e.g., conical. When a human passes into, through, or out of it, it triggers an event. The range and sensitivity of the trigger box can be adjusted

by developers. Also the class of detection can be changed from human to all objects, depending on the type of motion sensor being mimicked, PIR or ultrasonic respectively.

Presence - A presence sensor can be thought of as an ultra sensitive human motion detector (detecting even still people) which can also detect the identity of the detected people. Similar to the motion sensor, human entities can be detected and also identified by programmatically reading the identity of any people within the range of the sensor.

IMU - Each node within the game engine knows its own 3D location, rotation and acceleration information, updated at each game tick. Using this information a node can simulate an inertial measurement unit sensor.

Temperature - Unlike other sensors, the temperature sensor has no native primitive within the virtual world, as video games rarely model temperature. As a primitive, temperature could be modelled within a building by dividing the area into cuboids, matching the size of a room. As the temperature increases within a room, a connected space's temperature will also be influenced, at a speed dependent on the size of the temperature differential.

Light - A light actuator is connected to a light source in the virtual world, this could be a room light, spot light or lamp. The light source can be either a binary light or dimmable.

Sound - An sound actuator is connected to a sound source in the virtual world, such as an alarm or beep. When triggered the sound is played continuously until turned off.

Independent of the sensor type, either polling or event-based, data is published to the relevant sensor topic with the associated unique node ID, enabling other components to subscribe to these sensor updates. Whilst the event-based sensors only transmit sensor data upon being triggered by an event, such as a collision, or interaction, the polling sensors send sensor data updates at a fixed rate.

Virtual Environment & Physics

Within the 3D game engine developers can build virtual replicas of their target environments using the built in modelling tools, or import 3D models previously built, or from other CAD software, such as Blender[2] and 3DS Max. This allows developers to leverage existing designs

and models, or create new ones from scratch and share them with others e.g., an office block benchmark.

Within these built environments, the game engine is responsible for controlling a nodes location, utilising the physics engine to dictate how and where a node moves within an environment, e.g., falling due to gravity, reacting to collisions with solid objects such as walls and people, etc.

AI People and Crowds

Using the game engine's built in tools, developers are able to create and place virtual people into the environment. When creating people developers can specify their appearance and other physiological attributes, including their physical model, height, weight, walking and running speeds. Each of these attributes can have a significant effect, as they would in the real-world, on how the virtual person moves and interacts with the environment, e.g., reducing a person's size can enable them to fit through smaller spaces, or increasing their weight and speed will increase the force impacted on other objects upon collision.

6.4 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.

6.5 Mobility

Compared to traditional simulators, such as Tossim[32], Cooja[34] and NS2[6], which facilitate mobility only through manual scripting, our co-simulation approach, leverages the 3D game engine to fully realise dynamic and realistic human and device mobility within virtual environments.

Within the game engine it is possible to create both static and mobile devices, allowing developers to specify whether or not external forces can impact upon a nodes position, such as gravity or collisions from other objects. If static, devices stay fixed in place at the point they were deployed, it's possible to adjust this position before running simulations, but once running the devices will remain stationary; this can be useful when mobile nodes aren't needed and ensures unexpected movement won't cause any issues.

When setting a device as mobile, the device act as if it would in the real world. The device has a weight attribute which affects how it reacts to collisions with other objects. As the device moves it will collide with other solid objects, such as walls, people and other devices. Devices can also be attached to other mobile objects, such as people or robots, creating truly mobile nodes, that can be used to mimic mobile devices in the real-world.

As described in the previous section, developers can create and place virtual people within their environments of all different sizes, shapes, weights, speeds, etc. For integrating human mobility into a simulation, developers can utilise the AI toolkit provided by the Unreal Engine. Human mobility can be programmed in C++ or the visual programming tool, blueprints, to design simple movement patterns, or complex and reactive AI behaviours.

For example, to create a mobility behaviour that instructs agents to move between a set of locations at random, we first create and place several target locations within our environment. Next, we write, in either C++ or blueprints, a function to choose a random target location and then issue a move-to command to that location. Upon receiving this command, the agent will utilise the Unreal Engine's built-in navigation mesh to dynamically calculate the shortest route

to that location.

Will show an example blueprint demoing how to create a simple move-to behaviour.

For our indoor example we created several basic behaviours, including:

Move to location One or all people move to a specified location by the shortest route.

Roam All people roam between rooms and the corridor, simulating basic movement within an office.

Evacuate All people attempt to leave the building via the closest exit.

Avoid People will attempt to avoid an individual or object, changing their path or moving out of the way as it approaches.

Follow/Herd People will ignore their previous behaviour and follow an individual.

6.6 Communication

For communicating data between co-simulation tools we use an asynchronous event-driven approach, as previously outlined in the requirements chapter **??**. Our event driven approach is actually realised using event-notification, event-based-state-transfer and event-sourcing techniques[21, 22] to create an asynchronous responsive and flexible co-simulation platform. Using Apache Kafka[38], a real-time stream-processing pub/sub system, each co-simulation tool publishes or subscribes to a set of topics, such as sensors, for updating sensor information; actuators, for updating actuation information; or people, for updating the locations of people.

Each topic will typically have only one publisher, a co-simulation component which is responsible for managing and updating information related to that topic; for example, the game engine is responsible for managing and updating device hardware information, such as location in the virtual world, and any sensor information, such as whether the motion sensor has been activated or is a button pressed. On the other hand, a topic may have many subscribers (zero, one or more), which rely on events published to that topic, for example, the Cooja simulator subscribes to the sensor topic for updates about a device's location which it then reflects in its network map to generate a new radio model.

To reduce inter-component coupling and reduce latency, we use an event-notification approach, requiring all components which control or manage an aspect of the co-simulation platform to publish to a known topic when any state related to it is updated by that component; topics and their related events can be seen in table 6.1. Using this approach we can avoid components from requiring knowledge about where their events of interest are coming from, enabling us to later swap out or add additional tools with minimal configuration changes. Similarly, this halves the round-trip-time (RTT) as a component does not need to issue a request for the event, i.e., remote procedure call (RPC) or request response, it is instead notified of the event as soon as it occurs.

To further reduce coupling and reliance on other components data-stores, we also take an event-based-state-transfer approach, requiring publishers to publish not only that the event occurred, but also an up-to-date snapshot of the state that changed. Upon receiving an event the component has all the event related data, thus, removes the need for subscribers to request any state information from the publishing component and it's data-store, further reducing any latency caused by an additional RPC call. However, this approach has the drawback of increased memory usage, requiring subscribers to keep up-to-date their own copy of any state received that is of interest.

Lastly, we implement event-sourcing, in which all changes to the state of the simulation are are published as events and recorded by the event stream, such that they can be used to roll-back or replay a simulation. Kafka durably persists events, writing them to a log on-disk, with a parameterisable sliding window retention period over the stream of events, before dropping old events to make space for new ones. On top of this, each component has a unique event stream token, pointing to the last event they have observed, which makes it possible for individual

components to traverse the event stream as they wish; as new components are added to the stream at run-time, they can choose to process all events in the stream from the beginning, or simply start from the latest event. This creates a future-compatible platform, enabling developers to plug-and-play different components at run-time, test different configurations and observe behaviour using a variety of tools without the need to re-run simulations.

Using event-sourcing also opens up the possibility to replay isolated parts of a simulation; for example, we could replay the sensor topic, using it to drive a new simulation using the recorded inputs.

Kafka Topic	Event	Data	Type	Publisher	Notes
Sensor	Location	x, y, z	double	Unreal Engine	
Sensor	Button	Pressed	bool	Unreal Engine	
Sensor	PIR	Activated	bool	Unreal Engine	
Actuator	LED	Brightness	uint	Cooja	
Actuator	Beeper	On/Off	bool	Cooja	
Radio	Transmission	Source Destinations Interfered Destinations Packet	addr addr[] addr[] byte[]	Cooja	
Agent	Location	x, y, z	double	Unreal Engine	
Agent	Activity	Standing, Walking, Running	enum	Unreal Engine	

Table 6.1: Topics available to publish and subscribe to using Kafka.

6.6.1 Schema

For the two components, the Unreal Engine 4 game engine and the Cooja WSN simulator, we have implemented plugins which publish and subscribe to the topics listed in table 6.1. Topics are based on sensor inputs, which drive the Cooja simulator; actuator commands, which drive the actuator devices in the Unreal Engine; and agent information, which provide contextual information regarding the behaviour of agents within the Unreal Engine.

Kafka is a schema-agnostic by design, simply carrying bytes from publishers to subscribers via topics, a durably persistent stream of events. Because of this we can group similar events together, benefiting from Kafka's total ordering guarantees, ensuring that when replayed, events within a topic will be ordered as received by the stream.

Hence, it's necessary to create a defined but flexible schema for different components to communicate and ensure data is received in a correct and valid format. Because the events may

contain different fields and types, the schema must be flexible, allowing fields to be omitted when not needed, removing the overhead of sending dummy or zero data. One option is to create a flexible schema using a text-based key-value schema such as JSON providing a human-readable and dynamic schema, however, for small and frequent payloads a text-based approach is hugely inefficient, requiring a text-based key for even the smallest payloads. JSON also requires runtime exception handling code to ensure the flexible and dynamically-typed schema doesn't crash the receiver, due to omitting, changing the types of or adding new fields.

```
enum MsgType : byte { LED = 1, LOCATION, RADIO, PIR, PAUSE, RESUME, SPEED_NORM,
SPEED_SLOW, SPEED_FAST, RADIO_DUTY, FIRE, TEMP, SMOKE }

struct Vec3 {
  x:float;
  y:float;
  z:float;
}

struct RadioDuty {
  radioOnRatio:double;
  radioTxRatio:double;
  radioRxRatio:double;
  radioInterferedRatio:double;
}

table Message {
  id:int;
  type:int;
  location:Vec3;
  node:[RadioDuty];
  rcvd:[int];
  led:[int];
}
```

Instead, the co-simulator uses a more efficient, forwards- and backwards-compatible binary serialising and de-serialising approach, Google Flatbuffers[24]. Flatbuffers serialises data efficiently by removing structural information from the buffer, requiring minimal overhead for schema information, as sending and receiving parties are required to agree on a schema ahead of time. Flatbuffers fields are strongly typed, so errors are handled at compile-time rather than requiring additional code for run-time checks. Unlike other binary serialisation protocols, such as Google Protobuffers[25], Thrift[1] or Cap'n'proto[3], which require an unpacking step on the receiving side before accessing the data i.e., one-copy read, flatbuffers requires no unpacking, resulting in

zero-copy reads and faster processing of received buffers, ideal for real-time systems.

6.7 Synchronisation

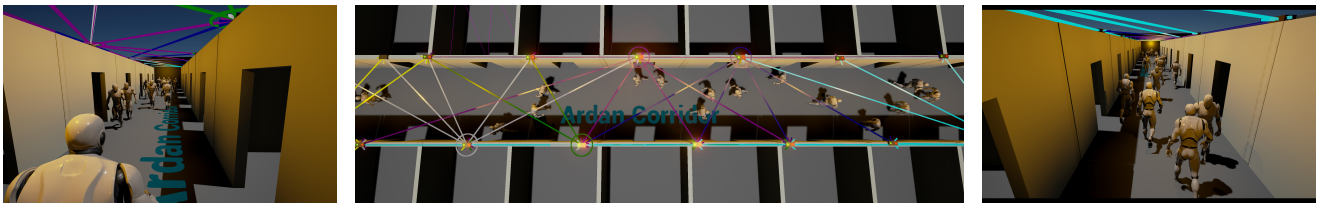
Due to simulation speed-ups or slow-downs, due to performance or scheduling issues, simulations can run ahead or behind their expected run-time, i.e., at 1x real-time speed, a simulation might experience brief performance slow-downs and only simulate 9.5 seconds in simulation time within a 10 seconds of real-time simulation, leaving a lag of 0.5 seconds. To keep the simulations synchronised between components, components inform each other of one another's simulated time via the time topic. Upon receiving a time event from another component, if its simulated time is out with its own by a set threshold, then it can pause to allow for the other component to catch up.

6.8 Forward Time Control

Using the Cooja simulator it's possible to run a simulation at a slower or faster than real-time rate, providing developers with more time to observe simulation or a faster completion time, respectively. Similarly, it's also possible to run a co-simulation in the same fashion, providing simulation speeds of 0.1x, 1x, 2x and 3x; which slows down or speeds up both the CPS simulation and virtual world simulation, enabling developers to run dynamic simulations with realistic physics in significantly less time than could be accomplished in the real-world.

In simulation pausing is also possible, giving developers the power to stop the world, observe, and analyse what activity is occurring in the system and world. Unlike a paused video recording, within the game engine, developers are able to explore the 3D environment whilst the world is frozen, providing developers with the ability to be omnipresent within the simulation. In cases where a cause-and-effect may not be obvious from one viewing angle, a developer can pause and investigate the world to assess the true cause of some phenomena from multiple angles; for example figure 6.5 shows the same moment from three angles, over the shoulder, birds eye and a security camera style view.

When a speed change is issued within the game engine by the user, the game engine publishes the change of speed to the time topic, alerting other components.



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

Figure 6.5: 15 people walking up and down the virtual corridor, triggering motion sensors

6.9 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.

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.

6.9.1 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.

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.

6.9.2 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 6.6, and developed further in figure 6.7, 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.

```
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 6.6: Lighting algorithm psuedo-code

```
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 6.7: Lighting algorithm psuedo-code with direction

6.9.3 "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 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

6.10 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 6.5. 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.

6.10.1 Co-simulator Performance

The results in figure 6.8 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.

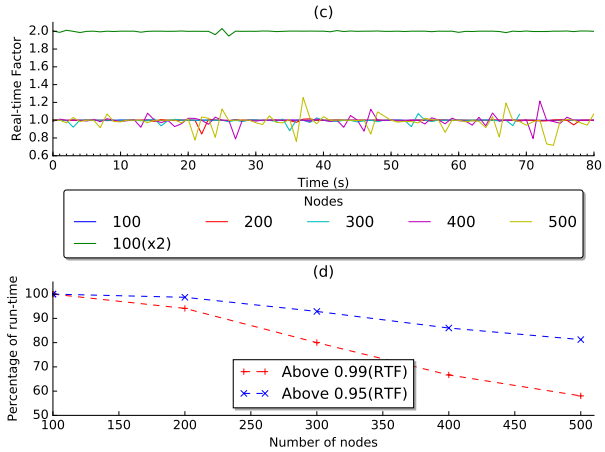


Figure 6.8: Figure (a) shows how the simulator speed fluctuates over time. Figure (b) shows the percentage of the total run time which the simulation maintains above 99% and 95% of its target speed.

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 of individuals. The results in figure 6.8 show that roughly half the number of nodes can be simulated in time with the game engine, with little fluctuation.

6.10.2 Faster-than-real-time Performance

6.11 Summary

Chapter 7

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

7.1 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

7.2 Visual Diffing

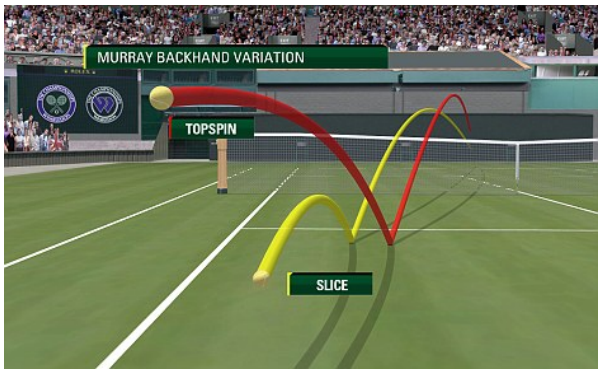
Understanding differences between two or more entities is a common but vital task across variety of domains, including text, image, voice and video analysis. System administrators or developers who need to know the differences between two text files employ the 'diff' UNIX tool[1], which highlights lines 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.

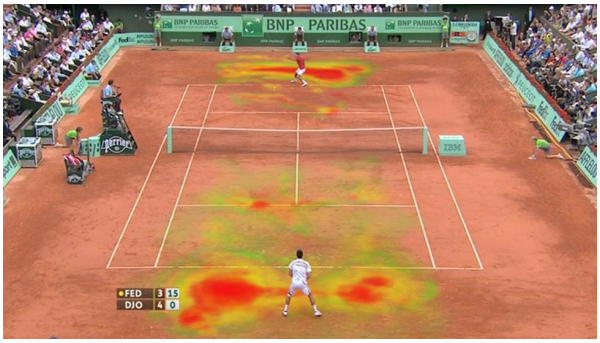
Whilst not formally described as visual diffing in other domains, techniques which can be described as such can be found in sports coverage and a variety of other media, shown in figure 7.1.

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

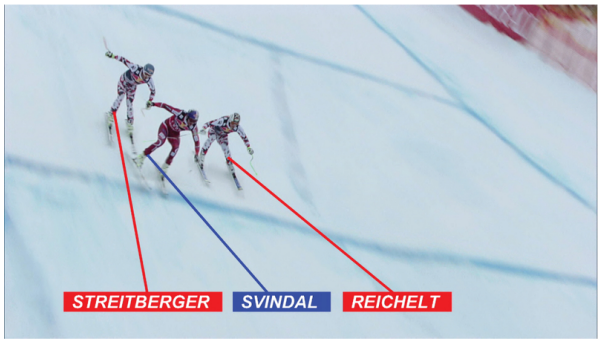
(a) The Hawk-Eye system showing where all service shots for a player land within the court.



(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 7.1: Examples of Visual Diffing within sports replays.

7.2.1 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:

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 7.1d; 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 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 7.1a 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 7.1c 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

7.3 Implementing Visual Diffing

Implementing visual diffing techniques utilises the recording framework described in section 7.4.

7.3.1 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

[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.

rendered as solids whilst the second's are ghosts, shown in figure 7.2.

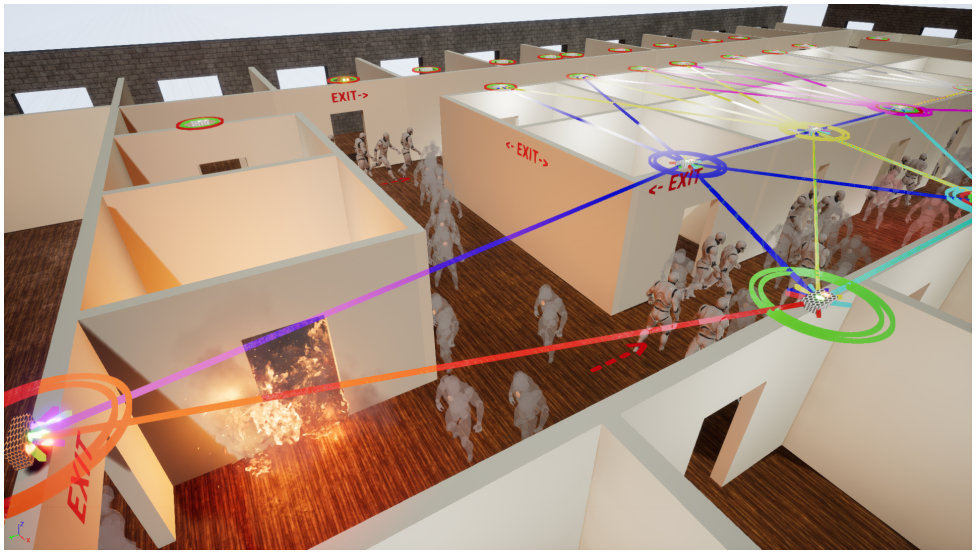


Figure 7.2: CPS-enhanced evacuation diffed with non-enhanced ghosts. Evacuees are being directed to exits via safe routes.

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.

7.3.2 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 7.3.

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 7.3.3.

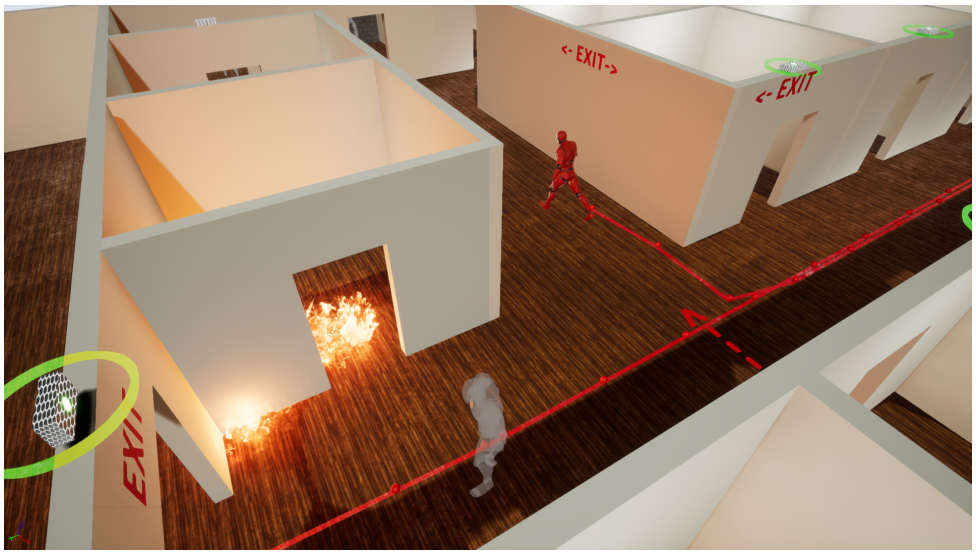


Figure 7.3: A person evacuating chooses two different paths in different runs based on CPS assistance, the ghost shows the same person taking a different route in a previous run.

7.3.3 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.

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.

7.3.4 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.

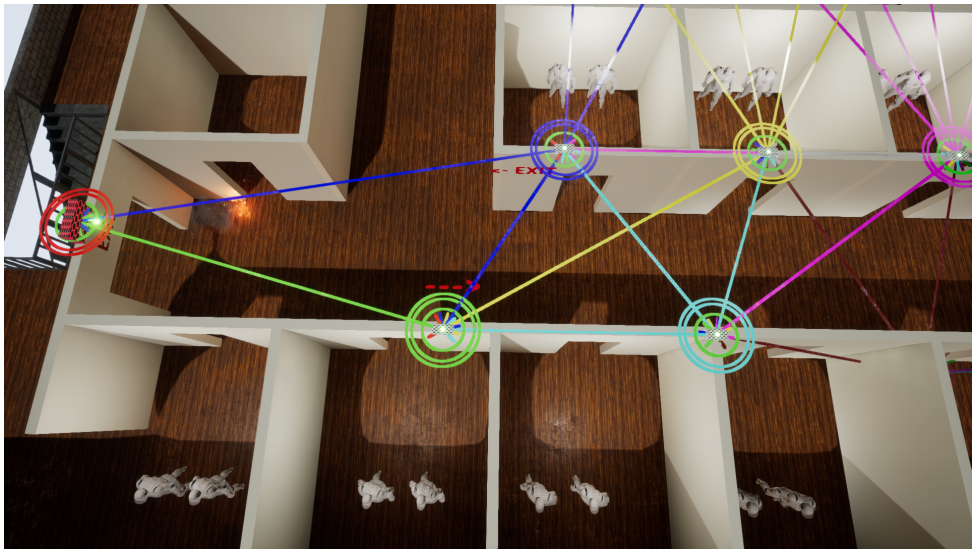


Figure 7.4: One of the sensors is red in colour and enlarged, representing a change in state when diffed to another run.

7.4 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 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.

7.4.1 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.

Table 7.1: 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

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 7.1.

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.

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 7.2: 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

7.4.2 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.

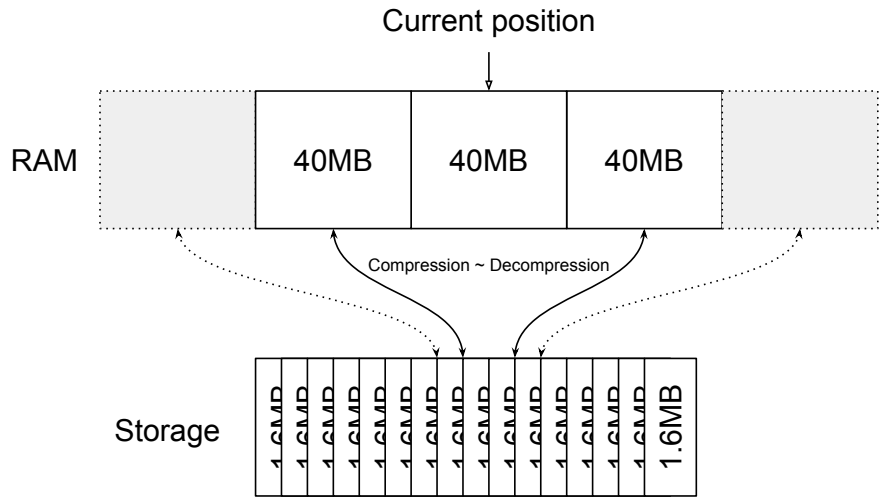


Figure 7.5: Segmented compression scheme employed by DejaVu to reduce in-memory usage.

7.4.3 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.

7.4.4 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.

7.5 Full Time Control

7.6 Using Visual Diffing

To demonstrate the visual diffing techniques implemented and described above, we created a case study for which we developed and tested a CPS application using these visual diffing techniques.

The remainder of this section first describes the case study and the related background, followed by a visual walk-through of using the visual diffing tools to test and analyse specific elements of the virtually deployed system.

7.6.1 Case Study: Fire Evacuation

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 [19, 12] has demonstrated the use and benefit of utilising a 2D evacuation simulator to develop new cyber-physical systems to enhance detection and evacuation, ensuring 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.

```
1 broadcast msg {ID, dist}
2 select:
3   waiton msg {ID, dist}:
4     routes[ID] = dist + 1
5     effectiveDist = min(dist+1, effectiveDist)
6     if changed:
7       broadcast msg {ID, effectiveDist}
8   waiton msg {ID, FIRE}:
9     routes[ID] = FIRE
10    effectiveDist = min_hazard(routes)
11    broadcast msg {ID, effectiveDist}
12  waiton sensor {FIRE}:
13    broadcast msg {ID, FIRE}
```

Figure 7.6: Pseudo-code for distributed evacuation CPS

The same algorithm runs on each node within the network, shown in pseudo-code form in figure 7.6. 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 7.2, 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 7.2, 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.

7.7 Evaluation

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

7.7.1 Compression

7.7.2 Real-time Performance

7.8 Conclusion

Chapter 8

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

8.1 Introduction

8.2 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?

8.3 A window into a CPS using AR

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

8.4 Tracking a CPS in the real world

8.5 Design

8.6 Case Study Analysis

8.7 Evaluation

8.7.1 Real-time tracking and analysis

8.7.2 Impact on CPS

8.8 Conclusion

Chapter 9

Conclusions and Future Work

9.1 Summary

9.2 Similarities, Limitations and Trade-offs

9.3 Summary of Thesis Achievements

Summary.

9.4 Applications

Applications.

9.5 Future Work

Future Work.

Bibliography

[1] Apache Thrift - Binary Serialisation Protocol (Apache Foundation).

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

[3] Cap'n'Proto Binary Serialisation Protocol.

[4] Carbon Trust: Energy Wastage Calculations.

[5] CryEngine 3D Game Engine. http://cryengine.com/.

[6] NS1/2/3 - Network Simulator. https://www.nsnam.org/.

[7] *The Regulatory Reform (Fire Safety) Order 2005, Part 2.* Queen's Printer of Acts of Parliament.

[8] The University of Cambridge Environment and Energy Green Challenge.

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

[10] UnrealEngine 3D Game Engine. http://www.unrealengine.com/.

[11] Long-Term Wireless Structural Health Monitoring of the Ferriby Road Bridge. *Journal of Bridge Engineering*, 15(2):153–159, 2010.

[12] O J Akinwande, H Bi, and E Gelenbe. Managing Crowds in Hazards With Dynamic Grouping. *IEEE Access*, 3:1060–1070, 2015.

[13] Tobias Baumgartner, Ioannis Chatzigiannakis, Maick Danckwardt, Christos Koninis, Alexander Kröller, Georgios Mylonas, Dennis Pfisterer, and Barry Porter. Virtualising Testbeds to Support Large-Scale Reconfigurable Experimental Facilities. In *Wireless Sensor Networks*, volume 5970, pages 210–223. Springer Berlin Heidelberg, 2010.

[14] H Bi and E Gelenbe. Routing diverse evacuees with Cognitive Packets. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, pages 291–296, mar 2014.

[15] Ann Blandford, Anne Adams, Simon Attfield, George Buchanan, Jeremy Gow, Stephann Makri, Jon Rimmer, and Claire Warwick. PRET A Reporter: evaluating Digital Libraries alone and in context.

[16] Ann Blandford, Anna L Cox, and Paul Cairns. Controlled experiments. In *Research Methods for Human Computer Interaction. CUP.*, pages 1–16. Cambridge University Press, 2008.

[17] Carlo Alberto Boano, Marco Zúñiga, James Brown, Utz Roedig, Chamath Keppitiyagama, and Kay Römer. TempLab: A Testbed Infrastructure to Study the Impact of Temperature on Wireless Sensor Networks. In *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, pages 95–106, Piscataway, NJ, USA, 2014. IEEE Press.

[18] Ioannis Chatzigiannakis, Stefan Fischer, Christos Koninis, Georgios Mylonas, and Dennis Pfisterer. WISEBED: An Open Large-Scale Wireless Sensor Network Testbed. In *Sensor Applications, Experimentation, and Logistics*, volume 29, pages 68–87. Springer Berlin Heidelberg, 2010.

[19] Nikolaos Dimakis, Avgoustinos Filippoupolitis, and Erol Gelenbe. Distributed Building Evacuation Simulator for Smart Emergency Management. *The Computer Journal*, 53(9):1384, 2010.

[20] Richard Figura, Matteo Ceriotti, Sascha Jungen, Sascha Hevelke, Tobias Hagemeier, Pedro Jo, and Mar On. Morpheus Simulate Reality for the Orchestration of Deployed Networked Embedded Systems. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2018.

[21] Martin Fowler. Event Sourcing, 2005.

[22] Martin Fowler. What do you mean by "Event-Driven"?, 2017.

[23] Andrea Gaglione, David Rodenas-Herraiz, Yu Jia, Sarfraz Nawaz, Emmanuelle Arroyo, Cecilia Mascolo, Kenichi Soga, and Ashwin A Seshia. Energy Neutral Operation of Vibration

Energy-Harvesting Sensor Networks for Bridge Applications. *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2018.

[24] Google. Google Flatbuffers.

[25] Google. Google Protocol Buffers (Protobuffers).

[26] Google. Google Tango Augmented Reality (ARCore).

[27] Dirk Helbing, Illés J Farkas, Péter Molnár, and Tamás Vicsek. Simulation of Pedestrian Crowds in Normal and Evacuation Situations.

[28] Chris W. Johnson and Louisa Nilsen-nygaard. Extending the Use of Evacuation Simulators to Support Counter Terrorism. In *The International Systems Safety Conference*, 2008.

[29] Akis Kokkinis, Aristodemos Paphitis, Loizos Kanaris, Charalampos Sergiou, and Stavros Stavrou. Demo: Physical and Network Layer Interconnection Module for Realistic Planning of IoT Sensor Networks. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2018.

[30] N Latman. TCPP Personality Profile. In *The Fourth Triennial International Fire and Cabin Safety Research Conference,*, Lisbon, Portugal, 2004.

[31] Lenovo. Lenovo Phab 2 Pro Google Tango-enabled AR Phone.

[32] 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.

[33] MathWorks. Simulink.

[34] 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.

[35] Fredrik Österlind, Joakim Eriksson, and Adam Dunkels. Cooja TimeLine: A Power Visualizer for Sensor Network Simulation. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 385–386, New York, NY, USA, 2010. ACM.

[36] Markus Schuß, Carlo Alberto Boano, Manuel Weber, and Kay Omer. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, page 336. Junction Publishing, 2017.

[37] Jonathan D. Sime. Affiliative behaviour during escape to building exits. *Journal of Environmental Psychology*, 3(1):21–41, mar 1983.

[38] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with Apache Kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, aug 2015.