Analogue-digital systems with modes of operation

Edwin Beggs\textsuperscript{1} and John V. Tucker\textsuperscript{2}

\textsuperscript{1} Department of Mathematics, College of Science, Swansea University, Singleton Park, Swansea, SA2 8PP, Wales, United Kingdom
e.j.beggs@swansea.ac.uk

\textsuperscript{2} Department of Computer Science, College of Science, Swansea University, Singleton Park, Swansea, SA2 8PP, Wales, United Kingdom
j.v.tucker@swansea.ac.uk

Abstract. We consider the nature of the relationship between physical environments, processes and systems, and the algorithms and software that monitor or govern them. In the case of complex physical situations, no one mathematical model can serve; rather their physical behaviour or operation may reveal several distinct regimes or modes. Thus, for the design of complex physical environments, processes and systems, a set of mathematical models – possibly of disparate kinds – may be needed, each model having its own domain of application and representing a single mode of behaviour or operation of the complex system. We propose a general definition of an analogue-digital system with modes based upon a portfolio of models, and we tackle the computational problem of transitioning from using one mode/model to another. We show that the modes/models can be bound together and transitions between modes/models can be visualised by a simplicial complex. We illustrate the ideas by reflecting on a simple example of driverless racing cars.

No plan survives contact with the enemy.

After Helmuth von Moltke the Elder (1800-1891)

There are known knowns; there are things we know that we know. There are known unknowns; that is to say, there are things that we now know we don’t know. But there are also unknown unknowns – there are things we do not know we don’t know.

Donald Rumsfeld (2002)

1 Introduction

A typical analogue-digital system is a system in which a continuous physical environment, process or component is monitored or governed by a discrete algorithmic process. This simple description covers an astonishingly large range of
systems, for example: classical control systems for machines, industrial plant, and buildings; networked devices that communicate data; monitoring and surveillance systems; and even scientific experiments, mediated by software, that measure physical quantities. Hybrid systems, embedded systems and, most recently, cyber-physical systems are types of analogue-digital systems whose applications are vast in scope and whose theories and design methods are maturing. We will raise and answer some foundational questions for a general theory of analogue-digital systems.

For simplicity, consider a real-world analogue-digital system that consists of physical equipment that is controlled by software on a processor. It is analogue-digital because the system processes continuous and discrete data: using real numbers to represent the data characterising the equipment, and integers and strings to represent the data processed by program.

Whilst the equipment is made from particular physical components, the controlling software is somewhat different. Its design is based upon

(a) dynamical systems that model the behaviour of physical equipment; and
(b) logical and algebraic models of the behaviour of the programs.

In the case of (a), the software and its underlying algorithms are designed to cope with any data that is input to and output from the physical equipment; their design depends on modelling the behaviour of the equipment by dynamical systems. In the case of (b), the behaviour of the programs is derived from the semantics of the specification and programming languages employed.

To create the software for an analogue-digital system the physical must be replaced by an abstract specification that documents certain operations and properties and that constitutes an interface between physical quantities and the algorithms and software. The interface must enable portability and verification so that the software can be certified “as reliable as possible” according to some best practice or engineering standard. For the designer of an analogue-digital system, the specification and validation presents certain problems. The reliability of an analogue-digital system depends upon the abstract assumptions about the physical system, and the sensors and actuators. These assumptions are at the heart of the dynamical systems that model physical reality.

A dynamical system is a mathematical model of an entity, process, environment, or phenomenon whose behaviour changes in time. The model represents behaviour by means of states that change over time. The model is likely to have parameters that are external inputs, which represent either

(i) external influences that are not predictable physical factors to which the system must respond or, conversely,

(ii) known commands or instructions designed to control the system.

The parameters usually involve real numbers. The state of a dynamical system can be based on continuous data, discrete data or a combination of both.

We propose to make a theory that can speak about three distinct components of a complex analogue-digital system: the physical system as it is in reality; the models of different aspects of the behaviour of the physical system; and the algorithms and software that employ the models to control the physical system.
Let us clarify these three distinctions by means of four principles.

**Principle 1: Physical uniqueness.** A physical system, process or environment is based on equipment made from particular physical components and its behaviour is a continuous physical phenomenon that is unique to that equipment.

**Principle 2: Physical specification.** A physical system, process or environment does what it does. To observe and measure a system is to abstract aspects of its behaviour in a way that is specific to the system. To model a system is to specify the system in an abstract and general way; the model specifies a class of particular physical systems.

The dynamical systems are, therefore, central to the study of the software. In addition to providing understanding of how the physical system might behave, they provide a specification of a general interface. But the models of a physical system may be many and varied; they are shaped by different choices of physical insights, spatial and temporal scales, and computational constraints, etc. There are many ways of modelling analytically (such as partial differential equations, ordinary differential equations) and computationally (such as finite elements, neural networks, lattices, cellular automata.)

**Principle 3: Modes and Models.** For a complex system, no one model is adequate but a portfolio of models, with overlapping fields of application, is needed to cover the system’s behaviour. Thus, a complex system possesses several models, each of which attempts to describe an aspect of its behaviour, which we call a mode of behaviour.

Whilst each mode of behaviour has its own individual model with a state space, the models can be disparate, and the system does not have a unified model or state space.

**Principle 4: Mode Transitions.** A methodology for managing a portfolio of models is needed that has methods for (i) evaluating the quality of the models in the portfolio; (ii) passing from one model to another; and (iii) dealing with exceptional situations when no model in the portfolio applies.

This paper is about how portfolios of models of physical components interface to algorithms and software; and how the algorithms and software may transfer between different models when suggested by the information available.

Our concept of an analogue-digital system consists of

- physical system + interface and protocol for data exchange + algorithm.

Specifically, we propose the algorithm treats the physical system as an oracle, whose queries are mediated by a protocol governing an interface.³ A good deal

³ The idea is a generalisation of Turing’s and Post’s ideas about oracles to algorithms in computability theory.
is known about the basic properties of such a model in cases where the physical systems are very simple and can be captured by a single model [4–9].

![Diagram of physical system and models](image)

**Figure 1:** Illustrating modes

We generalise this concept of an analogue-digital system by introducing modes represented by the different models. The technical question is:

*What conditions govern the transition from one mode/model to another: as the system operates one model becomes less relevant and another more relevant model must replace it for the software to function. How are the models bound or linked together?*

A complex addition to the interface and data exchange protocol is given based on *mode transition functions*. However, to get all of this to work, we propose a simple description of the geometry of the system, one which describes both the relationship between the various modes of operation of the system and when transitions between the modes are necessary. This suggests a topological space and we construct a simplicial complex – a space made of points, lines, triangles, tetrahedra etc., glued together using simple rules. The points represent the modes, and the lines or triangles joining them are used to show how well each mode describes the physical system. As this description is very explicit, it could be used for the design of the algorithmic structure of the system.

In Section 2, we introduce an example of a racing car under the control of software to clarify some intuitions of modes of operation and transitions between modes.

In Section 3, we give the general definitions of analogue-digital systems without and with modes, and introduce a simple software architecture to aid our discussion.

The next tasks are to show how: (i) to design modes; (ii) evaluate their fitness for purpose; and (iii) to change from one mode to another. In Sections 4 we address (i) and in Section 5 we address (ii). We show how to construct a simplicial complex to visually represent the mode transition protocol; this is inspired by the topological idea of a *nerve*. In Section 6, the transition theory is developed to address (iii).
In Section 7, we give a general mathematical example that characterises many classical systems based on manifolds and geodesics. These can illustrate modes but in the simpler case of the presence of some global state space.

In Section 8 we describe the first of two examples, one based on the informal car example in Section 2. In Section 9, we describe an example based on the orbits of planets.

In our concluding remarks, Section 10, we touch very briefly on two topics raised by considering the geometry of an analogue-digital system. The first concerns networks of analogue-digital systems; the second is about what happens when we look at the geometry of the system in finer or coarser detail. This is a form of multi-scale analysis that, in terms of topology, corresponds to a refinement of the modes or a presheaf structure.

We thank Felix Costa (Lisbon) for many enjoyable and influential conversations on analogue-digital systems; this paper has grown out of our collaboration on [4–9].

2 The problem of changing modes

We begin by exploring, informally, the need for modes of operation and transitions between modes using an example of driverless racing cars.

2.1 Example of a system with many modes: a robotic racing car

Consider a robotic car racing championship. The programmers of a racing car are given a list of the sensors and actuators, and a codification of their behaviour (e.g., the maximum acceleration is $10\text{m/sec}^2$; the time delay on the speedometer reading is $\frac{1}{2}\text{sec}$; ...). They are given a list of objectives (e.g., minimise the lap time; don’t get too close to the other cars or the edge of the track; ...).

Now the programmers separate concerns; the behaviour of the physical system is split into regimes, chosen to simplify the mathematical modelling and to program the objectives. These regimes are the basis of modes of operation. For our racing car, we might manage with five modes as follows:

*Mode $\alpha$: The car is on the track and some distance from the nearest cars.*

The state of any other car is reduced simply to a point position on the track, as it is not particularly relevant. The important factors are taking corners at the fastest safe speed, maximising acceleration and longer term factors, such as when to do a wheel change.

It would be nice to stay in mode $\alpha$ all the time, but this is unlikely as at some point the car will “meet” other cars. This can be determined from the picture available from the state space of mode $\alpha$ by looking at the distance to the nearest cars. When this distance becomes smaller than a certain value, the car will need to change behaviour. This change could be expected and predicted by the model, or it could be unexpected (e.g., the car hits a pool of oil and skids).
Mode $\beta$: The car is on the track but close to other cars.

The priorities are to avoid collision, and to overtake if possible. Now lots more state variables are needed to handle the nearby cars. For example, to the positions of the other cars, their velocities, accelerations and sizes are added. Also, some history or intelligence may be added (e.g., car A always tried to block us when we tried to overtake, but the car B did not).

At a particular combination of positions and velocities of nearby cars, a collision may be flagged up as a real danger.

Mode $\gamma$: The car is on the track but possibly on a crash course with another car.

The priority becomes not only to see if it is possible to avoid a collision, but to mitigate the effects if it cannot be avoided. At this point the actual geometry of the cars may be important, if say a wheel on wheel impact was likely to be more dangerous than a side on side impact.

These are three simple modes that make a nice initial design of a system; and maybe it could win a race, but there are gaps. Suppose the robotic car comes to a complete stop ending up off the track, undamaged on the grass verge; the programmers need to bolt on another mode:

Mode $\delta$: The car is off the track. The priority is to rejoin the track as soon as possible, but to stay away from cars on the track.

Experience generates further problems that must be considered. Suppose the speedometer develops a fault and core data is untrustworthy. As the software was not designed to cope with incorrect readings from the speedometer (rather than an obvious failure), a crash could result. To fix this the programmers need to add another mode:

Mode $\epsilon$: The car is on the track but the speedometer is faulty. Slow down and avoid other cars. Make way to the grass verge and stop.

To add this mode, the programmers need to carry out some modifications to the previous ones. These modifications are not major, they just need to say when an error mode $\epsilon$ should be entered. A comparison of the speedometer reading is made with the changes of positions in time of objects on the side of the track. If there is an obvious disagreement, we go to mode $\epsilon$. This change was made to $\alpha$ and $\beta$, as if we are in mode $\gamma$, we already have more urgent problems.

### 2.2 A modular approach to modelling complicated physical systems

From the point of view of designing an analogue-digital system, such as a driverless racing car, the modes represent a form of physical modularisation, determined by the science of the physical domain and its mathematical models. Consider two aspects of physical modelling.

**Modularity.** Mathematical models of physical behaviour are approximations, and they have limited ranges of validity. In complex environments and systems, physical conditions change, and the range of validity of a model can easily be
stressed and broken. Thus, \textit{the change of conditions requires a change of model}. In this way complex systems have different regions or \textit{modes}, where not only the equations governing the dynamics may change, but the entire description of the system may change, and a completely new set of state variables and a new type of model is used. The modes initiate a modular approach to modelling the system so that

(i) \textit{Specification}. We can keep track of properties that lead to a change of mode and model;

(ii) \textit{Implementation}. When a model is modified in a minor way, or more special cases added, we can isolate and limit the changes – and consequent errors – in software; and

(iii) \textit{Parallelism}. We can detect if a problem can be solved faster with parallel processing.

The models may be of disparate kinds: they may be analytical models based on PDEs or coupled systems of ODEs; or they may be algorithmic models based on numerical approximation methods, neural networks, coupled map lattices, and cellular automata. In a complex system one can expect them to be combined. There are methods of combining algorithmic models in different domains because algorithmic models are discrete space, discrete time models and can be unified in a theory of synchronous concurrent algorithms [27]; for example, in the case of whole heart modelling, see [18, 19, 11].

\textbf{Accuracy.} In a simple physical system, with easy access to measurements, it may be reasonable to assume that the software has instant access to a comprehensive specification of the physical system at any time. However, in practice, the system may be complicated enough to require some factors to be estimated. It would be helpful to have a modular approach to modelling the system that can cope with factors such as:

(i) \textit{Delay}. Measurements take time to perform, or be available only to a limited accuracy. There may be a time lag in obtaining some information that requires action by the software.

(ii) \textit{Decay}. Sensors, actuators and communications links are all prone to degradation (or even complete failure) and replacement will take time, if it is possible at all. If the performance of a component has degraded, the software may have to spot this from inconsistencies in the data, and take appropriate action.

(ii) \textit{History}. In there is no recent data to base a decision on, historical data may be needed to make a best guess.

To summarise modularity: every attempt must be made to reduce the complexity of the mathematical model, but the accuracy and ability to deal with a wide range of circumstances should not be compromised. To summarise accuracy: not only may the software’s idea of reality be only an approximation to reality, but it may be a misleading approximation.\footnote{The consequences of unreliable approximation to reality can be catastrophic, for example, on an airliner crash in the Atlantic following bad weather:}
3  Analogue-digital systems

We define a general analogue-digital system without modes and then extend the definition by introducing modes.

3.1 What is a general analogue-digital system?

The basic architecture of an analogue-digital system is depicted in Figure 2, in which arrows are used to indicate the direction of flow of information.

![Figure 2: A analogue-digital system](image)

The *physical system* is an environment, process or system that we wish to monitor or control. Suppose there is a fixed system of sensors for measurements, and actuators for control, linked to the physical system; so we should think of information flow along a *sensor bus* (e.g., from a thermometer) and an *actuator bus* (e.g., to a motor).

The *interface* is what translates between the sensors and actuators and the decision making software. The interface can take many forms; in a very simple case of the direct control of a motor by a microprocessor, it would be little more than digital-to-analogue (DA) and analogue-to-digital (AD) converters. The interface may need to manage error margins and timing delays. The response to a query normally takes time and it is possible that the response is timed out.

A *query* is a request from the decision making software to the interface and a *response* is a message going the other way. In the motor and microprocessor example, the query is simply a binary number interpreted by the interface’s DA-converter as a voltage applied to a motor, and there is no response.

Finally, there is the *algorithm* that formalises the decision making that attempts to monitor or control the physical system. The data must be considered separately, as it is the sole basis by which the algorithm knows what is happening in the physical system, it forms the algorithm’s picture of physical reality.

---

Temporary inconsistency between the measured airspeeds, likely following the obstruction of the Pitot probes by ice crystals that led in particular to autopilot disconnection and a reconfiguration to alternate law. – Final Report On the accident on 1st June 2009, Bureau d’Enquêtes et d’Analyses pour la sécurité de l’aviation civile
data must be updated, either periodically by standing orders in the interface, or when the algorithm sends a query to update items.

The remaining components of Figure 1, mentioned in italics, are the basis of specifications. The laws or theories refer to mathematical models of the physical system under consideration, e.g., Newton’s laws of motion or characteristics of electronic devices. Using these laws, and the sensors and actuators, an axiomatic specification of the interface is written.\(^5\)

The axiomatic description of the interface is used to write an algorithm to satisfy the list of objectives.\(^6\) For given initial configurations, if it can be proved that the algorithm suffices to guarantee the objectives then the algorithm can be said to be verified assuming the validity of the axioms. If it is not possible to guarantee that the objectives are always satisfied, a probabilistic approach may be used.

### 3.2 Oracles

The central idea is that the physical environment or system is an oracle to the algorithm. The algorithm asks questions and receives answers about the physical environment or system; specifically, the algorithm requests and receives data that it uses in its processing. The interface between the analogue and digital is of central theoretical interest.\(^7\)

To clarify the concept, imagine analogue-digital system as follows. Some physical device is rigged with sensors and actuators. The input and output leads are taken to an interface, where they are physically connected to a computer. (Picture the device in one room, the computer in another, with cables passing through a wall.) Thus, we have in mind a physical oracle, in which the computer may query the interface, and receive messages back. The ‘queries’ may be of various forms, e.g., interrogating a temperature sensor, or an instruction for a motor to be turned on. The algorithm need not know anything about the detailed operation of the sensors and actuators, that is all abstracted in the interface.

The idea of oracle is very general and is not confined to technological systems. For example, a complicated socio-technical example would be a laboratory service in a hospital. Instructions are received from doctors to perform tests, and the results are sent back to the doctors. There is a division of labour: the laboratory staff do not need to know about diagnosing disease, and the doctor does not need to know how to perform the tests. The different times taken for tests, the queue of requests and the capacity of the lab mean that the lab staff have

---

\(^5\) For example, such axioms might be:

- “System variable \( x \) can be measured to accuracy \( \epsilon \) in time \( t \).”
- “System variable \( y \) obeys the differential equation to within error \( \epsilon \).”
- “System variable \( z \) always remains in the interval \([-\pi, +\pi]\).”

\(^6\) For example, such an objective might be: “System variable \( x \) should always satisfy \( x \leq 30 \).”

\(^7\) The interfaces are far more complicated that the traditional oracles of computability theory. Since Emil Post’s development of computability theory, a query asks if a datum is in a set, and the response is yes or no and takes one time step.
to make scheduling decisions. If a request is urgent, they may have to decide be-
tween a quick but less reliable method and a slower more reliable test, of course
notifying the doctor of the possible errors concerned. In this example, a query by
a doctor might be “measure levels of X in patient Y by 09.00 tomorrow”, and a
response might be “level is Z, less accurate method used due to urgency of test”. 
Notice the prominent role of time in this example. Indeed, time can be expected
to be prominent in any service. The use of oracles may suggest a methodology
for designing and analysing a variety of systems.

3.3 Designing an analogue-digital system with modes of operation

Consider the process of designing an analogue-digital system whose behaviour
is split into *modes*. The aims are to (i) create the modes; (ii) develop the ob-
tjectives that make up their specification; and (iii) implement the objectives in
software. Starting with Figure 2, we imagine an architecture for three modes
\( \alpha, \beta, \gamma \) depicted in Figure 3:

![Figure 3: A three mode analogue-digital system](image)

Domain scientists and engineers must classify the behaviour of the physical
situation and define regimes or *modes*. Let these be indexed by \( \mathcal{M} \). Each mode
has a model that is a reasonable mathematical description of the physical system;
the model defines a list of state variables and a state space. In each mode,
appropriate approximations may be made to keep the mathematical description
manageable, and the behaviour of the inputs and outputs are codified in an
axiomatic description of the behaviour of the physical oracle. The state space
of the whole physical system can be *imagined* by patching together these state
spaces.

The domain scientists and engineers, together with the programmers, must
create the list of objectives for each mode \( \alpha \in \mathcal{M} \). Starting with the regime’s
model state variables and behaviour, a *representation* \( \text{State}_\alpha \) of the system in
mode \( \alpha \) is built in software. This will contain the state variables for mode \( \alpha \), and
probably more, such as estimates of the errors of the state variables, and some
of the known properties of the system.
In addition to coding the models of mode $\alpha \in \mathcal{M}$, there must also be conditions that have to be monitored that, if satisfied, cause a change from mode $\alpha$ to some new mode $\beta$.

In the detailed implementation of the software, problems will arise that may require revising earlier stages.

### 3.4 The architecture of a modal system

The components needed to program the system are summarised in the architecture depicted in Figure 4. Compared with Figure 3, the physical system is omitted and, for simplicity, only two modes $\alpha$ and $\beta$, and one transition from mode $\alpha$ to mode $\beta$, are shown. The next three sections will be devoted to explaining the principles behind this choice of components and architecture.

**Figure 4:** Details of a two mode analogue-digital system

### 4 Designing a mode implementation

#### 4.1 The architecture of a mode

For each mode $\alpha \in \mathcal{M}$ we have the following components:

- **state$_\alpha$**: The data in this component constitutes mode $\alpha$’s picture of the physical system, the state space State$_\alpha$. It can be broadly divided into **history**, **observations**, **deductions** or **estimations**. Much of this data may be unused by control$_\alpha$, but more of it is likely to be used by monitor$_\alpha$ (more on this later). Any system variables are likely to be accompanied by estimates of their errors, and the time of last measurement.

- **interface$_\alpha$**: This is the only component of the program that communicates with the physical system. Its only outputs to the rest of the program are the values it observes and passes to the component state$_\alpha$. Its only input from the rest of the program lies in its **query list**, the list of tasks to be performed with the physical
system. Queries may entail requests for observations, or control instructions. The interface comes with a default list of tasks designed to keep the states in $\text{State}_\alpha$ up to date. The tasks may be assigned various priorities, e.g., the tuples

- $<$ urgent priority, apply brakes, single action $>$
- $<$ background, measure temperature, accuracy 1 degree, repeat hourly $>$
- $<$ standard, measure temp., acc. 1°, if previous older than 1 minute, single $>$

Here we see that some tasks may be more urgent than others, while some may be repeated instructions. They may come with conditions for the measurements to be made, or with additional information, such as a required accuracy.

Why is this rather indirect method used for communication? One answer is that the controlling algorithm is really not interested in the details of a particular measuring device. Should a sensor wear out and be replaced by one from a different manufacturer, only the interface would have to be updated.

$\text{control}_\alpha$: This component of mode $\alpha$ actually makes decisions about controlling the physical system. It reads data from components $\text{state}_\alpha$ and $\text{orders}_\alpha$, and writes to $\text{state}_\alpha$ (typically estimations) and the query list for $\text{interface}_\alpha$. It might have some ability to respond to observed problems (e.g., by increasing the frequency of making some observations).

$\text{orders}_\alpha$: While the response of $\text{control}_\alpha$ may be largely determined by the code, there may be some flexibility in its behaviour. This could be viewed as setting the overall strategy for mode $\alpha$: let $\text{Orders}_\alpha$ be the set of possible instructions for mode $\alpha$. A simple example would be temperature control of a building, where the inhabitants could set the desired temperature range, and this range would be put in $\text{Orders}_\alpha$.

### 4.2 Accuracy, consistency, and evolution

In an analogue system there are both measured and calculated data. Tracking errors due to measurement, approximation in calculation, time delays, or any other source is a vital to the system. Not only are errors a measure of how well mode $\alpha$ is modelling reality, they also can tell if another mode of operation might do a better job in current circumstances – changing modes is our primary problem. So how do we know that mode $\alpha$ is doing its job of modelling reality?

Run the model and measure reality at the same time? Not so simple, recalling Observation 1 in the Introduction. Imagine an analogue-digital system with a constant $z$ of the system. Initially we measure $z \in (0, 5)$, and some time later we measure $z \in (4, 7)$. In what interval do we now know $z$ to be? It is tempting to say $z \in (4, 5)$, but maybe ...

1. Wear. The sensor is degrading with time, and that an increase in the midpoint of the interval is part of a gradual drift?

2. Domain. The day of the second measurement was very hot, and the specification of the sensor says that it should be used at cooler temperatures; that caveat keeps the sensor engineers in the clear, but what about the measurement?

3. Calibration. The equipment was not recalibrated after a minor shock, and the constant $z$ got shifted.
Now imagine the complicating factors in (2) and (3) were not known because the remote system did not have the sensors to report them. Our deduction $z \in (4, 5)$ becomes somewhat questionable. We may understand the errors, but do we understand the errors of the errors?\footnote{Compare the idealised situation in computable analysis. Take a computable real number $y$, and suppose we calculate $y \in (0, 5)$. Some time later we calculate $y \in (4, 7)$, so we deduce that $y \in (4, 5)$. As we perform more calculations, the error bound to within which we know $y$ never increases.}

Consider a worse case of a function $f(t)$ of time in an analogue-digital system. We measure $f(0) \in (0, 5)$, and the equations of motion given by a theoretical analysis of the system predict $f(1) \in (-2, 4)$. We then measure $f(1) \in (3, 6)$. We have all the previous complications (1-3), plus

4. **Validity.** Are the equations of motion really valid, or did we go outside their range of validity, or their numerical stability, or are there unexpected unknown unknowns?

Surely we want the measured value to be inside the range of the computed value? Measure $g(0) \in (0, 2)$, calculate $g(1) \in (-450, 900)$ and subsequently measure $g(1) \in (2, 4)$. The huge error margin in the calculated value simply illustrates that it is hopelessly inaccurate. The value of all calculated extrapolations tends to garbage as time increases, it is just a matter of how quickly.

It is important to distinguish between two sorts of observed errors, i.e. errors that we know about: If we estimate that the temperature should be in the range $(23.1, 23.2)$ and it is observed to be in the range $(23.4, 23.5)$ then we have an inconsistency. However, if we only need to know the temperature to an accuracy of one degree, then the estimate is accurate. In the example of $g(t)$ in the last paragraph, we had an example of consistency and inaccuracy. Accuracy matters because being inaccuracy may cause a failure of control for the system. Inconsistency matters because it tells us that there is a disagreement between the predicted behaviour and the observed behaviour of the system, and thus that the predicted behaviour should be considered unreliable.

5  **Evaluating a mode implementation**

5.1  **The architecture of mode evaluation**

For each mode $\alpha \in \mathcal{M}$ we have the following component:

`monitor_\alpha`: This is the only component of mode $\alpha$ which sends information to the system supervisor. Its output is specified by a *mode evaluation function*

$$P_\alpha : \mathcal{M} \times \text{State}_\alpha \times \text{Orders}_\alpha \rightarrow [0, 1].$$

The number $P_\alpha(\beta, x, o) \in [0, 1]$ assesses how well mode $\beta \in \mathcal{M}$ could run the system, given the current picture $x \in \text{State}_\alpha$ of the system and the current orders $o \in \text{Orders}_\alpha$. The value of the function is normalised by two given values $0 < p_{\text{low}} < p_{\text{high}} < 1$ so that
$P_\alpha(\beta, x, o) = 0$ means mode $\beta$ is incompatible with the current state

$P_\alpha(\beta, x, o) < p_{\text{low}}$ means mode $\beta$ models the system, but not well

$P_\alpha(\beta, x, o) > p_{\text{low}}$ means mode $\beta$ models the system reasonably well

$P_\alpha(\beta, x, o) > p_{\text{high}}$ means mode $\beta$ models the system well.

Note that the function $P_\alpha$ is total. The component is responsible for assessing observed problems such as consistency and accuracy (Section 4.2), and to what extent the observed state of the system is consistent with the objectives (including the current orders $o \in \text{Orders}_\alpha$).

5.2 Simplicial complexes and the geometry of the modes

The physical system has been divided or factored into a set $\mathcal{M}$ of regimes or modes of operation. The modes have been chosen to simplify the mathematical description of the physical system, and to make it easier to assign objectives and program the system in software. The system is represented by a portfolio of models. Here we are concerned with the large scale behaviour of the system and, therefore, the interconnectedness of the modes.

For some modes $\alpha, \beta, \gamma \in \mathcal{M}$, $\alpha \cap \beta \cap \gamma$ might be empty or nonempty. This means, is it possible for the state of the physical system to be simultaneously in (say) all three modes $\alpha, \beta, \gamma$. From this intersection information, we will build a geometric object, called the nerve of $\mathcal{M}$ \cite{note1}.

**Nerve of a category.** A 0-simplex is a point or vertex. A 1-simplex is a line segment connecting two vertices. A simplicial complex is made of several simplices, so we can see that a simplicial complex consisting of 0-simplices and 1-simplices is just a graph. However, simplicial complexes generalise graphs by allowing higher dimensional constructions. A 2-simplex is a triangle bounded by three 1-simplices. Figure 5 shows an example of a simplicial complex consisting of one 2-simplex, ten 1-simplices and eight 0-simplices or vertices $v$ indexed by $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \theta, \phi\}$.

![Figure 5: A simplicial complex](image)

The shading indicates that $\alpha\beta\gamma$ is a 2-simplex bounded by the 1-simplices $\alpha\beta$, $\alpha\gamma$ and $\beta\gamma$. Note that there is not a 2-simplex $\alpha\delta\gamma$. The dimensions continue to increase by a 3-simplex being a tetrahedron (triangular based pyramid) bounded by four 2-simplex sides, etc. Simplicial complexes are used in topology to give a computable presentation of topological spaces \cite{note2}.

Coordinates for the simplicial complex are illustrated in Figure 6, which consists of one 2-simplex, four 1-simplices and four 0-simplices $\{\alpha, \beta, \gamma, \delta\}$ (the 2-simplex $\alpha\beta\gamma$ is not shaded for clarity).

\footnote{For the general idea of a nerve of a category rather than a cover by subsets, see \cite{note3}.
Analogue-digital systems with many modes of physical operation

Figure 6: Points in a simplicial complex

The 1-simplex $\gamma\delta$ consists of points $t_\gamma v_\gamma + t_\delta v_\delta$ for $t_\gamma, t_\delta \in [0,1]$ and $t_\gamma + t_\delta = 1$. We identify the ends of the interval with the vertices, so $1_3 v_\gamma + 0_3 v_\delta$ is identified with $v_\gamma$ and $0_2 v_\gamma + 1_2 v_\delta$ is identified with $v_\delta$.

The 2-simplex $\alpha\beta\gamma$ consists of points $t_\alpha v_\alpha + t_\beta v_\beta + t_\gamma v_\gamma$ for $t_\alpha, t_\beta, t_\gamma \in [0,1]$ and $t_\alpha + t_\beta + t_\gamma = 1$. We identify the sides of the 2-simplex triangle with the corresponding 1-simplex intervals, for example $1_4 v_\alpha + 2_4 v_\beta + 0_2 v_\gamma$ is identified with $1_3 v_\alpha + 2_3 v_\beta$ in the 1-simplex $\alpha\beta$.

Construction of nerve. The nerve $N(M)$ of $M$ is a simplicial complex defined as follows.

For every mode $\alpha \in M$, there is a point or vertex $v_\alpha$. For two different modes $\alpha, \beta \in M$ with $\alpha \cap \beta$ not empty there is a 1-simplex $\alpha\beta$.

For three different modes $\alpha, \beta, \gamma \in M$ with $\alpha \cap \beta \cap \gamma$ not empty there is a 2-simplex $\alpha\beta\gamma$, and so on. To summarise this,

Definition. The element $\sum_{\alpha \in M} t_\alpha v_\alpha$ is in the nerve $N(M)$ of set $M$ of modes if, and only if,

(i) each $t_\alpha \geq 0$ and $\sum_{\alpha \in M} t_\alpha = 1$, and

(ii) taking the subset of $\alpha \in M$ for which $t_\alpha > 0$, the intersection of all the modes in the subset is nonempty (i.e., the subset gives a simplex of appropriate size).

5.3 The mode evaluation and classification functions

How can there be uncertainty as to which mode the system is in? Near the boundaries between two subsets it would be common for there to be an overlap. Suppose the program is in mode $\alpha \in M$. The value of the mode evaluation function

$$P_\alpha : M \times \text{State}_\alpha \times \text{Orders}_\alpha \rightarrow [0,1]$$

specifies $P_\alpha(\beta, x, o)$ which is a measure of how well $x \in \text{State}_\alpha$ could be described by mode $\beta \in M$ given the orders $o \in \text{Orders}_\alpha$. We need two fixed values:

A value $p_{\text{low}} > 0$ so that if $P_\alpha(\beta, x, o) > p_{\text{low}}$ then we say that $x \in \text{State}_\alpha$ could be reasonably described by mode $\beta$ given the orders $o \in \text{Orders}_\alpha$.

This sort of coordinate system for a triangle should be familiar to users of the RGB colour system.
A value $p_{\text{high}} > p_{\text{low}}$ so that if $P_{\alpha}(\beta, x, o) > p_{\text{high}}$ then we say that $x \in \text{State}_\alpha$ should be well described by mode $\beta$ given the orders $o \in \text{Orders}_\alpha$.

Conventionally we set $P_{\alpha}(\beta, x, o) = 0$ if mode $\beta$ is completely incompatible with the current state.

Now we use the nerve to make a mode classification function

$$N_\alpha : \text{State}_\alpha \times \text{Orders}_\alpha \rightarrow N(\mathcal{M})$$

as follows:

$$N_\alpha(x, o) = \frac{\sum_{\beta : P_{\alpha}(\beta, x, o) > p_{\text{low}}} (P_{\alpha}(\beta, x, o) - p_{\text{low}}) v_\beta}{\sum_{\beta : P_{\alpha}(\beta, x, o) > p_{\text{low}}} (P_{\alpha}(\beta, x, o) - p_{\text{low}})}.$$  \hspace{1cm} (1)

This function sends the current state and orders to the geometric picture of the state space as a simplicial complex with the modes as vertices.

Note that we only sum over modes $\beta$ for which $P_{\alpha}(\beta, x, o) > p_{\text{low}}$, i.e., we only consider those modes which have been certified as being reasonable descriptions of the system, given its observed state. Of course, an objective of the design of the system is that for any possible state of the system there is a mode that describes it reasonably well, and so that there is always a consistent choice of such modes.

As Figure 7 illustrates, there is no unified model for the system; the nerve binds the models together.

**5.4 Exceptions: known unknowns and unknown unknowns**

What could possibly go wrong? The answer is many things could go wrong. It is useful to consider Rumsfeld’s classification of unknowns. In our situation, and in system design generally, there are precise definitions:
Definition. A known unknown is an event that we knew might happen, that we knew the approximate form of the event, and, specifically, that the event has been written into the code. An unknown unknown is any unexpected event that we do not have the code to handle.

For example, in a robotic car race, another car may unexpectedly join the race after it has started; a designer can include a provision for adding cars during the race, which just requires some details to be added at the appropriate time. An automated system cannot run on the expectation that a team of programmers will upgrade its code at the last minute to deal with an unforeseen circumstance, it must make the best decisions it can.

To begin, how do we know when an unknown unknown is happening? The two exceptions that can occur with formula (1) might give a good indication of an observed unknown unknown:

(a) Partiality. The set of modes $\beta$ with $P_{\alpha}(\beta, x, o) > p_{\text{low}}$ may be empty, i.e., there are no modes which, in the opinion of the designers, model the current state reasonably well. Since we assumed $p_{\text{low}} \neq 0$, we have some room in which we may choose the best of a lot of bad choices. This is discussed further in Subsection 6.4.

(b) Contradiction. For the modes in Figure 5, a value of $N_{\delta}(x, o)$ of $\frac{1}{2}v_{\gamma} + \frac{1}{2}v_{\delta}$ is fine, as there is a line connecting the vertices $v_{\gamma}$ and $v_{\delta}$, so the designers knew that those modes could in principle be consistent at a given time. However $\frac{1}{2}v_{\beta} + \frac{1}{2}v_{\delta}$ is an invalid value, it does not lie in the simplicial complex. The designers never considered the possibility that these modes $v_{\beta}$ and $v_{\delta}$ could both be valid descriptions of the same physical state. We have a possible contradiction – this is discussed further in Subsection 6.5.

6 Making transitions between modes

6.1 The architecture of mode transition

There are two components that accomplish a change of mode, one of which was not illustrated earlier in Figure 3:

supervisor: This component decides on the change of mode from (say) mode $\alpha$ to mode $\beta$. Its input is the value of the mode evaluation function $P_{\alpha}(\beta, x, o)$ from monitor$_{\alpha}$. It assigns the mode $\beta$ to control the system by (i) changing state information State$_{\alpha}$, and (ii) modifying that mode’s order list Orders$_{\alpha}$. In the case of observed unforeseen circumstances (Sections 6.5 and 6.4) it must decide on the system’s response. The component is the only part of the program which receives commands from outside agencies; for example, it is responsible for initialising the whole system on startup; and it can receive an over-ride to move to an emergency shutdown mode.

transfer$_{\alpha}$: This component moves information between modes – changing states – and implements the mode transition functions

$$\tau_{\beta\alpha} : \text{State}_{\alpha} \times \text{Orders}_{\alpha} \rightarrow \text{State}_{\beta} \times \text{Orders}_{\beta}$$
It is the only part of mode $\alpha$ which needs to know anything about State$_\beta$. The function $\tau_{\beta\alpha}$ is called by the supervisor, and when it is computed control is transferred to mode $\beta$.

### 6.2 The mode transition functions $\tau_{\beta\alpha}$

If we decide to move from mode $\alpha$ to mode $\beta$, how do we actually implement this? Each mode $\alpha$ carries a picture State$_\alpha$ of the physical system, so to move from $\alpha$ to $\beta$ we need a mode transition function $\tau_{\beta\alpha}$, so that our picture of reality is redrawn in a form understood by mode $\beta$. However, this function need not be defined on all of State$_\alpha$; in fact the designers are only likely to have implemented the function $\tau_{\beta\alpha}(x)$ on the condition that $\beta$ is a reasonable description for $x \in \text{State}_\alpha$.

The transition functions also depend on the standing orders, Orders$_\alpha$. For example, when controlling temperature, we might simply store the target temperature of a building in the standing orders, e.g., Orders$_\alpha = \{21^\circ\text{C}\}$. In calculating the mode evaluation function $P_\alpha$ we obviously need to take account of the target temperature, but why is it needed in $\tau_{\beta\alpha}$? Simply copying the standing orders, so we would get Orders$_\beta = \{21^\circ\text{C}\}$ may negate the point of changing: the target temperature could be modified by various factors, such as the time of day or the cost of energy, giving e.g., Orders$_\beta = \{19^\circ\text{C}\}$.

Thus, we need the mode transition function to have the form

$$\tau_{\beta\alpha} : \{(x,o) \in \text{State}_\alpha \times \text{Orders}_\alpha : P_\alpha(\beta, x, o) > p_{\text{low}}\} \rightarrow \text{State}_\beta \times \text{Orders}_\beta.$$  \hspace{1em} (2)

We need to assume that the designers have done a sufficiently good job of implementing $\tau_{\beta\alpha}$ so that the system can carry on from this position in mode $\beta$. If this proves too difficult to do, then it is possible that $\beta$ is not a reasonable description for all $(x,o) \in \text{State}_\alpha \times \text{Orders}_\alpha$ with $P_\alpha(\beta, x, o) > p_{\text{low}}$, and that the function $P_\alpha$ may have to be redefined.

Note that not all the state variables in State$_\beta$ need be set by the mode transition function in (2), some may be left as $\text{undefined}$. To use the example of the car race in Section 2.1, if we are in mode $\alpha$ we may decide to switch to mode $\beta$ because another car is within a certain distance. However there is no reason why we needed to have much information on that particular car before now. In particular we may have no idea of its velocity, or its identity, so these may be left as $\text{undefined}$ after the switch in modes. It is to be hoped that the designers of mode $\beta$ have given a high priority to finding a value for these $\text{undefined}$ variables.

One problem with changing modes is what happens when it is done several times in quick succession. Suppose that we have just used the function in (2) to move from mode $\alpha$ to $\beta$. It may happen that we almost immediately switch to mode $\gamma$. However, the designers could only be expected to have taken account of this possibility if $\gamma$ is a reasonable description of $\tau_{\beta\alpha}(x) \in \text{State}_\beta$. In that case, we get the composition

$$\tau_{\gamma\beta} \circ \tau_{\beta\alpha} : \{y \in \text{State}_\alpha \times \text{Orders}_\alpha : P_\alpha(\beta, y) > p_{\text{low}} \text{ and } P_\beta(\gamma, \tau_{\beta\alpha}(y)) > p_{\text{low}}\}$$
However, we might have moved from $\alpha$ to $\gamma$ in one go, using

$$\tau_{\gamma\alpha} : \{ y \in \text{State}_\alpha \times \text{Orders}_\alpha : P_\alpha(\gamma, y) > p_{\text{low}} \} \rightarrow \text{State}_\gamma \times \text{Orders}_\gamma.$$

As we could have taken either path from $\alpha$ to $\gamma$, we should hope that there is a reasonable consistency between $\tau_{\gamma\beta} \circ \tau_{\beta\alpha}$ and $\tau_{\gamma\alpha}$ as functions on the subset of $\text{State}_\alpha \times \text{Orders}_\alpha$ where all three inequalities in (3) and (4) are satisfied, and this should be part of the design requirements.

### 6.3 Normal running

Changing modes changes the way a system is viewed and may be needed to simplify and frame a computational description – it is likely to be a necessary evil. Changing the way a system is viewed comes with a cost: change takes time, it may introduce inaccuracy or inconsistency, and it may lose information. For a robust many mode system, a method is needed by which the supervisor minimises the changes of mode. Periodically, the program checks that it is in a correct mode to describe the system, as follows:

If we are in mode $\alpha$, and $P_\alpha(\alpha, x, o) > p_{\text{high}}$, then $x \in \text{State}_\alpha$ together with the standing orders $o \in \text{Orders}_\alpha$ should be well described by mode $\alpha$. We therefore remain in mode $\alpha$.

If $P_\alpha(\alpha, x) \leq p_{\text{high}}$, we calculate the other values of $P_\alpha(\beta, x, o)$ and use (1) to find $N_\alpha(x)$. We could then select the $\beta \in \mathcal{M}$ which gives the largest value of $P_\alpha(\beta, x, o)$. We could modify this in the event of several large values by using the more detailed geometry of the symplectic space by choosing a vertex $v_\beta$ which $N_\alpha(x, o)$ is approaching, comparing the last several values of $N_\alpha(x, o)$. In this manner we pick a mode which the state is entering, and so might be expected to stay in for longer.

This suggest a plan: cover the possible states and standing orders with modes, in such a manner that there is always a mode that models the state well (i.e., with mode evaluation function $> p_{\text{high}}$). Under this assumption we can show that every mode does its job properly, and that we do not have to change modes too often.

However, as von Moltke observed, ‘no plan survives contact with the enemy’: our enemy is the physical reality of the system. We need to know when the plan is going wrong, and to do. In summary, the plan fails but the real world continues to move on and decisions have to be taken.

We examine the two obvious exceptions which can be thrown by the mode transition function $N_\alpha$. These are listed at the end of Section 5.3, and we now examine each in more detail.

### 6.4 Partiality exception: Into the unknown

Corresponding to partiality exception (a) in Section 5.3 we have the problem:
\{ \beta \in \mathcal{M} : P_\alpha(\beta, x, o) > p_{low} \} = \emptyset.

In other words, for all \beta we have \( P_\alpha(\beta, x, o) \leq p_{low} \), i.e., the software does not think that any mode \( \beta \) reasonably describes the picture and orders \((x, o)\).

The assumption, in Section 5.2, that \( p_{low} > 0 \) means we have some ‘wiggle room’, a gap between where ‘reasonable description’ ends and ‘complete garbage’ begins. We can still try to choose the best of a lot of bad choices – we may take the largest value of \( P_\alpha(\beta, x, o) \) even if it is \( \leq p_{low} \). Essentially we may have a mode \( \beta \) which the engineers thought was some sort of description of the system, but they were not prepared to certify as a ‘reasonable description’. When using such a mode \( \beta \) to control the system, it may be wise to have a ‘safe strategy’ for \( \beta \) – a version of Orders_\beta written on the understanding that mode \( \beta \), while the best available, may still not be a reliable representation of the real world.

But which \( \beta \) do we choose? It is possible that the values of \( P_\alpha(\beta, x, o) \) may not distinguish well between values in the range \( \leq p_{low} \) – after all, the system was never supposed to be run in that range. When lost but with a last known position mode \( \alpha \), we can bias the search for new modes towards “nearby” modes. The simplicial complex provides an idea of the distance between modes: the shortest path in terms of number of edges or 1-simplices between the vertices.

For example, if for \( \alpha, \beta \in \mathcal{M} \) we have a \( \gamma \) with \( \alpha \cap \gamma \neq \emptyset \) and \( \gamma \cap \beta \neq \emptyset \) but \( \alpha \cap \beta = \emptyset \) then \( d(\alpha, \beta) = 2 \). Alternatively, if the collection of modes \( \mathcal{M} \) is part of a multi-scale hierarchy (Section 10.2), we might move to a coarser collection of modes.

If the system does not swiftly return to a known mode, and the system has not collapsed, a warning would be flashing, alerting people to intervene. Of course, this intervention might be another automated process rather than human.\(^{11}\)

### 6.5 Contradiction exception

Corresponding to the contradiction exception (b) in Section 5.3 we have the problem:

\( \{ \beta \in \mathcal{M} : P_\alpha(\beta, x, o) > p_{low} \} \neq \emptyset \) but has no common intersection.

In the case of three modes \( \beta, \gamma, \delta \) with \( P_\alpha(\beta, x, o) > p_{low}, P_\alpha(\gamma, x, o) > p_{low} \) and \( P_\alpha(\delta, x, o) > p_{low} \), the following two configurations of intersections of modes in Figure 8 both give problems:

\[\begin{array}{c}
\begin{tikzpicture}
\node (alpha) at (0,0) {$\beta$};
\node (beta) at (-1,-1) {$\gamma$};
\node (gamma) at (-2,-2) {$\delta$};
\end{tikzpicture}
\end{array}\]

\[\begin{array}{c}
\begin{tikzpicture}
\node (alpha) at (0,0) {$\beta$};
\node (beta) at (-1,-1) {$\gamma$};
\node (gamma) at (-2,-2) {$\delta$};
\end{tikzpicture}
\end{array}\]

\(^{11}\) The required abstract understanding may well be beyond current artificial intelligence; see [15] for progress on this front.
We have statements that \((x, o) \in \text{State}_\alpha \times \text{Orders}_\alpha\) could be reasonably described by a collection of modes which (according to the engineers) has no common intersection. But surely this is a real contradiction (or the engineers are mistaken), as \((x, o)\) would be in the intersection? The answer is no, and the reason is lack of information or errors. Remember that the point of modes is that \(\text{State}_\alpha\) is not the state space of the whole system, but rather a picture created by the software to describe a given aspect of the system. A scenario may illustrate this:

**Example: Aircraft.** A military aircraft spots a new contact on its radar screen. There is, as yet, insufficient information to say if it is a hostile military plane or a civilian plane. Both possibilities could reasonably describe the situation as either might well be true or false; but, in particular, there is no possibility that describes the situation well. If we just have the two disjoint modes \(\alpha = \text{hostile}\) and \(\beta = \text{civilian}\) to choose from, we have the case where there is more than one possible mode assignment consistent with the data (not necessarily a problem), but the assignments are, in reality, mutually exclusive (a problem). The reason is lack of data. We know that the designers did not consider this case, as if so they would have drawn a line joining \(\alpha\) and \(\beta\) in the simplicial complex, and no exception would have occurred. Of course, it might be said that the designers should have considered this case, and that would be a valid point to make at a committee of enquiry into any disaster resulting from the situation, but is not much help at the time.

So we know there is a problem, but what do we do? We can flag an error, and hope that a human puts the matter right before something unfortunate happens – if the system can simply be shut down in a safe mode. Extra information could resolve the problem, and the exception should trigger several queries for more information. But what do we do now? Here are four options.

We could assign the most likely mode, in the knowledge that it may well be wrong (1). The most likely mode could be used, but with a “safe strategy” (already mentioned in Section 6.4) which is used when the mode assignment is considered unreliable (2). We could have a “fail safe” order on the functions \(P_\alpha\) (3). We could have a consensus strategy between the modes (4).

**Example: Aircraft.** Let us see how these strategies might work in the airplane scenario:

1. Hawk. Knowing that there is a significant chance it might be wrong, the software takes the largest \(P_\alpha\), and, e.g., as a result fires missiles to shoot the target down. For example, in the absence of other information, the matter was decided because the unknown plane was not on a commercial flight path.

2. Dove. Following on from (1) the plane may assume that the other aircraft is hostile \(\alpha\), but it is operating under standing orders used when the mode is active but considered to be unreliable. In this case an obvious thing to write in \(\text{Orders}_\alpha\) would be ‘do not fire missiles’.

3. Fail-safe. Written into the code for \(P_\alpha\) is the guarantee:
\[ P_{\alpha}(\text{civilian}, x, o) > p_{\text{low}} \implies P_{\alpha}(\text{hostile}, x, o) < P_{\alpha}(\text{civilian}, x, o). \]

This means that a taking the largest choice defaults to civilian rather than hostile, in the case where the situation is reasonably described by the unknown plane being civilian. Thus, if we fail to identify the plane successfully, the default is fail-safe (well, as far as the possible civilians are concerned). Of course there is a slight inconsistency here – if the engineers rigged this fail-safe, why did they not simply include code for handling an intersection of hostile and civilian?

4. Fail-safe: Consensus. During the design process the two design teams for the modes are asked: ‘Suppose that we have an inconsistent mode allocation, but the evidence indicates that your mode may be applicable. What orders should be issued as a result, given that your mode may equally well not be applicable?’ The civilian mode designers might reply ‘do not fire missiles’, and the hostile mode designers might reply ‘take evasive action to avoid attack’. Now whichever mode takes control of the system is given consensus orders containing both the orders above. This is effectively a way of making up some sort of fail-safe strategy as the design develops.

7 Systems modelled by manifolds and geodesics

We illustrate the concepts we have introduced using models of physical systems based on differential manifolds [20]. A manifold is a topological space whose neighbourhoods have local coordinate systems based on \( \mathbb{R}^n \). These local coordinate systems are called charts. If a manifold captures the state space of a model of complex system then the (i) charts can be used to define modes and (ii) the behaviour of the system model in time is a path through the manifold passing through the charts. Such mathematical models of systems are very common. They are, however, a special case as our theory of multi-modal analogue digital systems is specifically designed so as not require there to be a global state space for the system.

7.1 Smooth manifolds

A topological space \( X \) is an \( n \)-dimensional smooth manifold if it possess a family of charts. Let \( \mathcal{M} \) be some index set. According to the usual definition, a chart consists of

\[
\text{open subsets } U_{\alpha} \subset X \text{ and homeomorphisms } \phi_{\alpha} : U_{\alpha} \to V_{\alpha},
\]

where each \( V_{\alpha} \) is an open subset of \( \mathbb{R}^n \) and \( \alpha \in \mathcal{M} \).

Furthermore, for every \( \alpha, \beta \in \mathcal{M} \) with \( U_{\alpha} \cap U_{\beta} \) not empty, the transition function

\[
\tau_{\beta \alpha} = \phi_{\beta} \phi_{\alpha}^{-1} : (\phi_{\alpha}(U_{\alpha} \cap U_{\beta}) \subset V_{\alpha}) \to (\phi_{\beta}(U_{\alpha} \cap U_{\beta}) \subset V_{\beta})
\]
is a smooth function (i.e., differentiable arbitrarily many times). Further if \( U_\alpha \cap U_\beta \cap U_\gamma \) is not empty we have the equality

\[
\tau_{\gamma \alpha} = \tau_{\gamma \beta} \tau_{\beta \alpha} : (\phi_\alpha(U_\alpha \cap U_\beta \cap U_\gamma) \subset V_\alpha) \to (\phi_\gamma(U_\alpha \cap U_\beta \cap U_\gamma) \subset V_\gamma). \tag{6}
\]

To compute with this we need a function to tell us which chart we are in. This is given by a partition of unity [22] p. 40, which is a collection of functions:

\[
\theta_\alpha : X \to [0,1] \text{ with } \sum_\alpha \theta_\alpha(x) = 1 \text{ for all } x \in X \text{ and } \theta_\alpha(x) = 0 \text{ if } x \notin U_\alpha.
\]

(Thus we have a ‘part’ of ‘unity’ 1 for each open set.) We relate these functions to the charts \( V_\alpha \subset \mathbb{R}^n \) by defining

\[
P_\alpha(\beta) = \theta_\beta \phi_\alpha^{-1} : V_\alpha \to [0, \infty). \tag{7}
\]

For purposes of computation it is convenient to extend this to a function on domain \( \mathbb{R}^n \) by taking the function to be zero outside \( V_\alpha \). If there are originally \( m \) open sets, then if we choose \( p_{\text{low}} = \frac{1}{m+1} \), for any \( \alpha \) and \( y \in V_\alpha \) we are guaranteed to have at least one \( \beta \) so that \( P_\alpha(\beta, y) > p_{\text{low}} \).

To find a better bound for \( p_{\text{low}} \) in terms of the dimension, the reader can read about the Čech-Lebesgue covering dimension [14].

The theory of manifolds is extensive and contains many intriguing results about inequivalent families of charts and the embedding of \( n \)-dimensional manifolds in sufficiently higher dimension spaces \( \mathbb{R}^N \). The question as to whether and how computation on manifolds should be split into coordinate charts – rather than, e.g., regarding it as a subset of a higher dimensional Euclidean space – is answered in the extensive and closely related literature on the choice of grids for global climate models, e.g., [2].

### 7.2 A dynamical system

Consider how a practical problem involves the idea of a manifold. The manifold represents numerically and geometrically the space of all possible states of a system. The behaviour of the system in time is represented – numerically and geometrically – by a path in the manifold. There is a differential equation governing the time evolution of the coordinates of states which are points in the path traversing the manifold. Periodically, we have to change coordinate charts.

For example, consider the idea of a geodesic or minimum distance path on a curved surface; it has been likened to the path of an ant crawling on the surface of an apple [17]. In Einstein’s general theory of relativity [12] the paths of the planets are given by geodesics in a space-time manifold, and the resulting prediction of the perihelion precession of Mercury was central in convincing people about that theory. (See [17, 21] for the mathematical details.)

The definition of a manifold and geodesic motion fits our idea of computation of many modes as follows. To begin, we assume that we have a register machine with real number registers, and operations that enable it

(i) to calculate precisely all those inconvenient transition functions between charts; and even
(ii) to solve the differential equations for the geodesics. The regimes or modes are exactly the set of charts indexed by \( \mathcal{M} \). The picture of the system corresponding to each mode \( \alpha \in \mathcal{M} \) is \( \text{State}_\alpha = \mathbb{R}^n \times \mathbb{R}^n \). We consider \( (y(t), \dot{y}(t)) \in \mathbb{R}^n \times \mathbb{R}^n \), where \( y(t) \) is the state (e.g., position of our ant, airplane or planet) at time \( t \) in the coordinate chart \( V_\alpha \), and \( \dot{y}(t) \) is its rate of change or velocity. No attempt to place any restriction forcing \( y(t) \in V_\alpha \) is needed: it is the job of the functions \( P_\alpha(\beta, (q, p)) = \theta_\beta \phi_\alpha^{-1}(q) \) to ensure that charts are changed as appropriate. (This type of example does not include Orders\(_\alpha\) as the program only has one purpose, with no variability.)

Next, we consider how these functions are computed. In computable analysis, computable reals and computable functions of computable reals are computed by programs generating rational numbers with error bars [28]. To find the position at a given time to a given error, we have to find how accurately we need to know the initial point, and then time step into the future, keeping track of errors and (rather messily) changing charts where we have to. There may be singularities, places beyond which we cannot extend the geodesics, but bearing that in mind, this is a perfectly consistent computable mathematical theory. However, we no longer have \( \text{State}_\alpha = \mathbb{R}^n \times \mathbb{R}^n \), we need a more complicated data type incorporating error bounds for \((y(t), \dot{y}(t))\). (For analogue-digital systems see [13].)

What complications can arise between our mathematical model and the observations?

1. Overdue computation. The real physical system waits for no computer, it evolves regardless of whether the computer has finished a particular calculation. In computable analysis, if the velocity becomes too large we can reduce the time-step in the program to compensate. In real physical time we may not have this option, and the velocity may become so large that we even lose track of which chart or mode the system is in.

2. Chaotic computation. Many physical systems are chaotic, effectively meaning that even if we had an arbitrarily large amount of time to compute and an exact physical theory, we still could not measure the initial conditions exactly enough to extrapolate beyond a certain time limit. Of course, this could be a known unknown, and we can force the system to compensate by regularly observing the most chaotic variables, and using these to correct the calculated values. In the case of the solar system, this would be the positions of the planets around their orbits, as opposed to the much more slowly changing orbital parameters (for the solar system see [16]).

8 **A racing track**

8.1 **A single car**

Consider a toy car race around the following track:
Figure 9: A racing track

Mode $\alpha$: For a single car, the slots on the track keeps the car moving in a given path, and all we have to worry about is getting the fastest time round the track, taking account of the problem that the maximum speed of the car is too fast for the semicircular parts of the track, so the car has to slow down there. For car $i$ this mode $\alpha_i$ has State $\alpha_i = (q_i, v_i)$, where $q_i$ is the position of car $i$ and $v_i$ is the velocity of car $i$.

Mode $\beta$: When two cars 1 and 2 are on the track, they can collide where the track narrows at the chicane, at position $q_c$. To stop this happening, a new mode $\beta_i$ is introduced for car $i$ for collision avoidance. We need data on both cars, so State $\beta_i = (q_1, v_1, q_2, v_2)$.

Mode evaluation: For simplicity we assume that mode $\beta_i$ is activated any time that car $i$ is near the chicane. We have mode evaluation functions

$$
P_i^\alpha(\alpha, (q_i, v_i)) = f(|q_i - q_c|),
$$
$$
P_i^\beta(\beta, (q_i, v_i)) = 1 - f(|q_i - q_c|),
$$
$$
P_i^\beta(\alpha, (q_1, v_1, q_2, v_2)) = f(|q_i - q_c|),
$$
$$
P_i^\beta(\beta, (q_1, v_1, q_2, v_2)) = 1 - f(|q_i - q_c|),
$$

where $|q_i - q_c|$ is the distance from car $i$ to the chicane, $d$ is chosen according to the stopping distance of the car, and $f$ is given by the graph in Figure 8:

Figure 10: The graph of $f$
**The nerve:** This consists of the 0-simplices $\alpha$ and $\beta$, and the 1-simplex $(\alpha, \beta)$.

![Diagram of nerve](image)

**Figure 11:** Single car nerve for the car race

**Transition functions:** We suppose that Orders is trivial for simplicity. Then $\tau^1_{\alpha,\beta} : \text{State}_{\alpha} \rightarrow \text{State}_{\beta}$ is given by $\tau^1_{\alpha,\beta}(q_1, v_1, q_2, v_2) = (q_1, v_1)$ and $\tau^1_{\beta,\alpha} : \text{State}_{\alpha} \rightarrow \text{State}_{\beta}$ is given by $\tau^1_{\beta,\alpha}(q_1, v_1) = (q_1, v_1, ?, ?)$. Here ? denotes an undefined value, and one which it is a high priority to find out about.

### 8.2 Several cars

Implementing the safety measures in the car race depends on knowing the positions of the other cars. This is likely to involve other processors, as competing teams would likely be unhappy at the thought of a single process controlling both team’s cars. In other words, we have a network. Just how much cooperation between nodes of the network do we need to ensure that our safety measures work? That depends...

**Case A:** A team fitted their car with a camera to observe nearby cars. They can now implement mode $\beta$ entirely independently of what any other team does.

**Case B:** The race organisers have made a condition of entry to the competition, that all cars must make their position public if they are near the chicane, in order to avoid accidents. If car 1 enters mode $\beta^1$ it queries car 2, and if car 2 is also in mode $\beta^2$ it is obliged to respond.

**Case C:** A particular team has incorporated into mode $\beta$ software to maximise their team points in the competition, in the case that two of its cars are racing each other. For example, one car could let the other always go first into the chicane. If possible, the team might actually combine the processes $\beta^1$ and $\beta^2$ into a single process $\beta^{12}$ running on a single computer.

### 9 A solar system

#### 9.1 Real time modelling of a solar system

The Sun, Moon and Earth form a three body gravitational system (more if we include the other planets), and that there is no exact general solution known to the three body gravitational problem. The point is that we do not have a general three body problem with the Sun, Moon and Earth: To a very good
analogue-digital systems with many modes of physical operation

approximation the Moon orbits the Earth, and the Earth-Moon system orbits the Sun, and our computational problem is vastly simplified.

More generally, take an example of a solar system with three planets \(\{1, 2, 3\}\) orbiting a star. As long as the planets are a reasonable distance apart, the dominant feature in their orbits is the star’s gravity, with minor influences from the other planets. This is modelled by processes or modes \(\{\alpha^1, \alpha^2, \alpha^3\}\), which can easily be run on different processors or nodes, as occasional communication of the other planets orbits is sufficient to keep the planet \(i\)’s orbit updated.

\textbf{Modes}: Let us concentrate on planet 1. If one of the other planets, say 2, comes within a certain distance of 1, then we require much more detailed knowledge of the position and velocity of planet 2 to accurately model the path of planet 1. We will call this mode \(\beta_{12}^1\) (reserving the superscript \(i\) for a process run on node \(i\)). Likewise there is a mode \(\beta_{13}^1\) used when planet 3 is close to planet 1, and a mode \(\gamma_{23}^1\) when both other planets are close to 1. Thus node 1 has four modes of operation, \(M^1 = \{\alpha^1, \beta_{12}^1, \beta_{13}^1, \gamma_{23}^1\}\).

\textbf{Mode evaluation}: To choose between these modes, we use indicator functions \(\{w_{12}^1, w_{13}^1\}\). The definition of these is quite simple, the value \(w_{1i}^1 \in [0, 1]\) and \(w_{1i}^1 \approx 0\) when planet \(i\) is ‘far’ from planet 1, and \(w_{1i}^1 \approx 1\) when planet \(i\) is ‘close’ to planet 1. Thus \(w_{1i}^1\) may be thought of as a fuzzy logic truth value of the statement \(planet \ i \ is \ close \ to \ planet \ 1\).

\begin{equation}
(0, 0, 0, 0) \quad (0, 0, 1, 0)
\end{equation}

\begin{equation}
(1, 0, 0, 0) \quad (0, 1, 0, 0)
\end{equation}

\begin{equation}
(0, 0, 0, 1) \quad (0, 0, 0, 1)
\end{equation}

\begin{equation}
(1, 0, 0, 0) \quad (0, 1, 0, 0)
\end{equation}

\begin{equation}
(0, 0, 0, 1) \quad (0, 0, 0, 1)
\end{equation}

\textbf{Figure 12}: Configurations of planets

The indicator functions \(\{w_{12}^1, w_{13}^1\}\) are converted to mode evaluation functions by

\begin{equation}
(w_{12}^1, w_{13}^1) \mapsto (1 - w_{13}^1)(1 - w_{12}^1), w_{12}^1 (1 - w_{13}^1), (1 - w_{12}^1) w_{13}^1, w_{12}^1 w_{13}^1),
\end{equation}

where we use the order \(\{\alpha^1, \beta_{2}^2, \beta_{3}^3, \gamma_{23}^1\}\). In Figure 12 we have a configuration of planets with, say as indicated by the dot on the square of \(w_{23}^1, w_{31}^1\) values, \((w_{23}^1, w_{31}^1) = (0.8, 0.2)\), giving mode evaluation functions \((0.16, 0.64, 0.04, 0.16)\). Thus the most suitable mode for node 1 in Figure 12 is \(\beta_{2}^3\). Figure 12 also shows the value of the mode evaluation functions for the corners of the square of \(w_{23}^1, w_{31}^1\) values.

\textbf{The nerve}: This is the solid tetrahedron with vertices \(\{\alpha^1, \beta_{2}^2, \beta_{3}^3, \gamma_{23}^1\}\). It is a 3-simplex because, as shown in the example, we expect that all four mode evaluation functions can be nonzero simultaneously.

\textbf{Transition functions}: We have not specified the data structures associated to the modes, however if the modes \(\beta\) used Einstein’s general relativity and mode \(\alpha\) used Newton’s theory, it might be imagined that quite a bit of data conversion would have to be done.
9.2 Cooperation between processes tracking planets

It was found that communication problems between different machines meant that running two separate processes for planets which were close together was too inaccurate (evidently this star system is more interesting and dangerous than ours). It was decided to instigate a cooperative process $M^{12} \subset M^1 \times M^2$, consisting of just one mode $\beta^{12} = (\beta_1, \beta_2)$ running on one machine. Thus when node 1 and note 2 agree that planets 1 and 2 are close, but that planet 3 is further away, an attempt will be made to start a single process $\beta^{12}$ to model the positions of planets 1 and 2. Should that attempt fail (for example a necessary communications link not be established), then the separate processes $\beta_1$ and $\beta_2$ will continue to run.

The cooperative process $\beta^{12}$ ends when we leave $M^{12} \subset M^1 \times M^2$. The process $\beta^{12}$ makes sure that the pictures State$_1$ and State$_2$ respectively are updated, and then switches control back to the individual nodes in modes $(\beta_1, \beta_2) \in M^1 \times M^2$. Given that $(\beta_1, \beta_2)$ is no longer the preferred mode, either node 1 will then instigate a mode transition, or node 2 will do so. The cooperative mode transfers control to $(\beta_2, \beta_1) \in M^1 \times M^2$ simply because it was not necessary to duplicate the machinery of the mode transition functions.

Of course, we should also have $M^{13} \subset M^1 \times M^3$ and $M^{23} \subset M^2 \times M^3$ operating on the same principle. But what happens if all three planets are close? Following from two node cooperations, we could have three node cooperations $M^{123} \subset M^1 \times M^2 \times M^3$, consisting of just one mode $\gamma^{123} = (\gamma_1, \gamma_2, \gamma_3)$.

10 Concluding remarks

We have encountered a range of phenomena that suggest further refinements; they raise classic questions about parallelism and hierarchy.

10.1 Networks and cooperation

The two examples of analogue-digital systems with many modes in Sections 8 and 9 illustrate levels of cooperation between analogue-digital systems: there are two cars and three bodies. In those cases the systems are identical, but this is not necessary. In a quarry we could have diggers with modes \{digging, moving\} and trucks with modes \{loading, unloading, moving\}.

Consider a general network of analogue digital systems and suppose that some of these systems can cooperate; the systems are nodes in the network. There are penalties for cooperation, such as communication speed limits between nodes of the network or problems with the increased complexity of the system. Designers need to decide under what circumstances to attempt cooperation, and how to proceed if an attempt fails. There has to be a good reason to cooperate between systems, say to speed and simplify the operation of the combined system.

Consider two analogue digital systems $AD^1$ and $AD^2$, with respective set of modes indexed by $M^1$ and $M^2$. For ease of explanation, suppose they are
Analogue-digital systems with many modes of physical operation 29

running on different machines. We suppose that these systems are in general unrelated, so that the set of modes for the combined system \( M \) contains the product of the modes for the subsystems \( M^1 \times M^2 \). If there is no cooperation between \( AD^1 \) and \( AD^2 \), we will simply have \( M^1 \times M^2 \), with \( M^1 \) being executed in parallel to \( M^2 \).

What would a cooperation between systems \( AD^1 \) and \( AD^2 \) look like? Suppose that the designers have identified modes \( \alpha^1 \in M^1 \) and \( \theta^2 \in M^2 \) as a possible cooperation. The default is that \( \alpha^1 \in M^1 \) and \( \theta^2 \in M^2 \) will still be executed in parallel.

However, if we are in \((\alpha^1, \theta^2) \in M^1 \times M^2\), an attempt will be made to start a cooperative mode \( \alpha\theta^{12} \). This new mode might consist simply of modified versions of \( \alpha^1 \in M^1 \) and \( \theta^2 \in M^2 \) which were optimised to run together, or at the other extreme it might consist of a completely new single process taking the place of both \( \alpha^1 \in M^1 \) and \( \theta^2 \in M^2 \) and running on a single machine.

10.2 A multi-scale hierarchy of modes

What roles do decomposition and modularity, encapsulation and inheritance, play in implementing multi mode analogue-digital systems? The behaviour of the physical system, and therefore the portfolio of models, have plenty of structure. Consider our racing car story in subsection 2.1:

Example: Robotic Race. Suppose the team decided that it would improve performance and safety to take account of the track conditions, specifically, if the racing track was dry or wet. As the code for mode \( \alpha \) (the car is on the track and some distance from the nearest cars) was already quite complicated, they did not add separate cases to it. Instead they split \( \alpha \) into two sub-modes, \( \alpha \)-dry and \( \alpha \)-wet. The picture State\( \alpha \)-dry = State\( \alpha \) and State\( \alpha \)-wet was State\( \alpha \) with an additional state variable for just how wet the track was. Now only part of the code for \( \alpha \) needed to be rewritten to give the codes for \( \alpha \)-dry and \( \alpha \)-wet.

Further, they decided that the old mode evaluation functions \( P_\alpha \) were working perfectly well, so they did not rewrite them. Only if mode \( \alpha \) was chosen did a further choice have to be made, between \( \alpha \)-dry and \( \alpha \)-wet. They also needed to decide if mode \( \alpha \) was to be an active mode, i.e., a mode that could control the system, or a purely passive mode whose only purpose was to choose between the sub-modes \( \alpha \)-dry and \( \alpha \)-wet and to calculate the evaluation functions. They decided that mode \( \alpha \) should be active, as it was a perfectly good compromise solution if for some reason neither \( \alpha \)-dry nor \( \alpha \)-wet could be chosen reliably.

The answer to the question of decomposition comes from topology. A refinement of a given open cover for a topological space is an open cover – i.e., collection of open subsets whose union is the whole space – and so that any subset is completely contained in some subset of the original coarser cover.

A refinement is like increasing the resolution in a picture: we will resolve modes into submodes, an idea to be taken geometrically literally if we compare the nerve of the cover and the nerve of its refinement.
The set of modes $\mathcal{M}'$ is a refinement of $\mathcal{M}$ if for every $\xi' \in \mathcal{M}'$ there is an $\alpha \in \mathcal{M}$ so that $\xi'$ is contained in $\alpha$. We take this containment to be both physical (the subset of the physical state space corresponding to $\xi'$ is a subset of that corresponding to $\alpha$), and virtual (the software for $\xi'$ is inherited from that for $\alpha$).

In our racing example above, the refinement $\mathcal{M}'$ of the original modes $\mathcal{M}$ of Section 2.1 contains $\alpha$-dry and $\alpha$-wet, and we have the containments $\alpha$-dry $\subset$ $\alpha$ and $\alpha$-wet $\subset$ $\alpha$.

One complication that arises is that of multiple inheritance (see e.g., [23]). If the $\xi' \in \mathcal{M}'$ above is also contained in $\beta \in \mathcal{M}$, then we also need $\xi'$ to inherit from $\beta$.

In fact what we have been describing resembles a generalisation of the manifold data type, the idea of pre-sheaf [10], where data is localised to subsets of a set. Given a collection of subsets of a set, a presheaf assigns a structure $\mathcal{P}(U)$ to every $U$ in the collection, together with a map (called the restriction map) $\phi_{VU} : \mathcal{P}(U) \to \mathcal{P}(V)$ for every $V \subset U$ in the collection. The basic idea is quite simple, given information about a set, we can restrict it to information about a subset.

The idea of refinement in Section 10.2 gives an implementation of the idea of presheaf, in that we have a restriction map from $\text{State}_\alpha$ to every $\text{State}_{\xi'}$ for every $\xi' \in \mathcal{M}'$ with $\xi' \subset \alpha$. Such a restriction map is used in going to finer scale structure in a multi-scale hierarchy. In our case we would also have to be able to move to coarser scales, which is not usually implemented in the structure of a presheaf. However one definition in sheaf theory might be useful: the idea of reconstructing data on a coarser resolution by combining finer resolutions. A sheaf is a special case of a presheaf where, given subsets $V_i \subset U$ where the union of the $V_i$ is all of $U$, then we can reconstruct $\mathcal{P}(U)$ from the collection of the $\mathcal{P}(V_i)$.

The connection between presheaves and databases and information theory has been noted in various places, e.g., [1, 26].

References

3. Alexandroff P, Über den allgemeinen Dimensionsbegriff und seine Beziehungen zur elementaren geometrischen Anschauung, Mathematische Annalen 98 (1928), 617-635.

12 There are several generalisations of presheaves, which eventually lead to just a functor between categories.