Visualising Interval-Based Simulations

Amadeusz Pawlik och Henry Andersson

Avancerad Objektorienterad Programmering

Halmstad 2015-06-16
VISUALISING INTERVAL-BASED SIMULATIONS

Henry Andersson & Amadeusz Pawlik

School of Information Technology, Halmstad University, Sweden, 2015
Acknowledgements

We would like to express our sincerest gratitude to the patient soul, our supervisor Adam Duracz, for the invaluable guidance, advice and support that he has given us.

We extend our thanks to Xu Fei for his assistance.

Thank you!
Amadeusz & Henry

Dedicated to our computers, with which we’ve spent many pleasant nights.
Contents

1 Introduction 5
  1.1 Purpose ................................................. 5
  1.2 Requirements ............................................. 6
  1.3 Scope .................................................. 6

2 Background 7
  2.1 Simulations .............................................. 7
  2.2 Acumen .................................................. 10
  2.3 Scala ..................................................... 11
  2.4 3D environment .......................................... 11
  2.5 Visualisation ............................................ 12
    2.5.1 Visualising results of simulations of hybrid systems .... 13
  2.6 Related work ............................................. 13

3 Method 15
  3.1 Approach ............................................... 15
  3.2 Requirements, delimitations and results ..................... 16
    3.2.1 Requirements analysis ................................ 16
    3.2.2 Requirements fulfilment plan ......................... 17
  3.3 Delimitations ............................................ 18
    3.3.1 Validation of results ................................ 18
    3.3.2 Testing ............................................. 18
  3.4 Milestones 1 & 2 - Acquiring relevant values by extraction .... 18
  3.5 Milestone 1 - Visualising intervals in an accessive way .... 19
    3.5.1 Assessment of the method ............................ 20
    3.5.2 Motivation .......................................... 20
  3.6 Milestone 2 - Visualising intervals in a rigorous way ....... 21
    3.6.1 Assessment of the method ............................ 22
    3.6.2 Motivation .......................................... 22

4 Results 23
  4.1 Visualisation ............................................ 23
  4.2 Source code ............................................. 23
  4.3 Run-time performance .................................... 24
  4.4 Test results ............................................. 25

5 Discussion 26
  5.1 Flaws .................................................. 26
VISUALISING INTERVAL-BASED SIMULATIONS

June 16, 2015

Abstract

Acumen is a language and tool for modeling and simulating cyber-physical[2] systems. It allows the user to conduct simulations using a technique called rigorous simulation that produces results with explicit error bounds, expressed as intervals. This feature can be useful when designing and testing systems where the reliability of results or taking uncertainty into account is important.

Unfortunately, analyzing these simulation results can be difficult, as Acumen supports only two ways of presenting them: raw data tables and 2D-plots. These views of the data make certain kinds of analysis cumbersome, such as understanding correlations between variables. This is especially true when the model in question is large. This project proposes a new way of visualising rigorous simulation results in Acumen.

The goal of this project is to create a method for visualising intervallic values in 3D, and implement it in Acumen. To achieve that, every span of values is represented as a series of overlapping objects. This family of objects, which constitutes an under-approximation of the true simulation result, is then wrapped inside a semi-translucent box that is a conservative over-approximation of the simulation result.

The resulting implementation makes for a combination of mathematical correctness (rigour), and mediation of intervals in question. It enables the user to explore the results of his rigorous simulations as conveniently as with the existing, non-rigorous simulation methods, using the 3D visualisation to simplify the study of real-life problems.

To our knowledge, no existing software features visualisation of interval-based simulation results, nor is there any convention for doing this. Some ways in which the proposed solution could be improved are suggested at the end of this report.
1 Introduction

In the process of designing a system it is advantageous to use a simulator. It allows the engineer to investigate a particular system’s behaviour in different scenarios without having to construct a prototype. Therefore, it is favourable to use simulation while working on systems that are expensive, systems that are of a theoretical nature and in larger projects where designs are developed collaboratively.

To conduct a simulation, a number of tools are needed - a modelling language, a simulator engine that carries the mathematical part of the simulation and a way to show the results for further examination.

Acumen[1] comprises all of these; a simulator for modelling and simulation of cyber-physical systems. Acumen currently allows the user to execute simulations in two different manners; using either floating-point numbers or enclosures (interval-valued functions). Depending on the desired application of the simulation and other possible requirements, both techniques exhibit certain benefits and drawbacks. Furthermore, simulation results can be examined in different ways: as a plot, raw values or in 3D.

The problem that Visualising Interval-Based Simulations is meant to tackle is the following: there is no convention for taking into account uncertainty resulting from interval-based simulations in the 3D visualisation of the corresponding model. Furthermore, no such method is available in Acumen. That is a serious issue for the user, as 3D-models are far easier to understand in comparison with plots and tables - especially when simulating large systems with many variables. Intervals or, more generally, enclosures arise as the result of over-approximative numerical methods when the simulator accounts for the uncertainty of real variables, numerical- and representational error. This uncertainty must be considered when mapping values to 3D.

To propose a way of dealing with the uncertain-value mapping-problem is an important step in making formal methods, such as interval-based simulation, more accessible and useful. Visualisation of uncertain values in 3D gives the user a better understanding of the ongoing relationships between variables, something that is difficult given the current data visualisation facilities. As Acumen is used for both educational and scientific purposes, e.g. modelling of robots and Advanced Driver Assistance Systems[17], enhancing Acumen would be favourable for both students and researchers as ”[... ] in the design process, the [...] models should be easy to understand and analyse. Moreover, it should be easy to deduce the [...] relationship between the models[... ]”[32].

1.1 Purpose

This project’s purpose is to create a method for representing uncertain values when visualising 3D Acumen models.

Currently, there is no method for visualising interval-based simulation results, as there
is no method for visualising intervals resulting from such simulations. A solution to this problem is proposed in the form of a method for visualising intervals, and that solution is implemented in Acumen in order to make 3D-visualisation of interval-based simulations available to the potential user.

1.2 Requirements

Models to be supported:

- a model with no uncertainty, with a linear differential equation,
- a model with uncertain initial values,
- a model with uncertain parameters,
- a larger model comprising multiple modules.

Features to be supported:

- Types of models to be supported: Box, Sphere, Cylinder, Cone, Custom meshes (objects)
- Parameters for object manipulation to be supported: center, size/radius/length, rotation

Other:

- All possible outcomes are to be rigorously represented in the visualisation.
- Uncertainty in every parameter and every possible combination of these parameters are to be represented in the visualisation.

1.3 Scope

- The intervals represented in the visualisation are to envelop the correct values; however, the representation does not have to exclusively envelop correct values (meaning it can be over-approximated).
- Acumen supports using Text in the 3D-visualisations. However, over-approximations of the translations, rotations and scalings of Text objects generally make them unreadable for the user. Therefore, Text objects are omitted from the 3D visualisation of interval-based simulations.
2 Background

The following research touches simulations, the technical background of the Acumen simulator itself, the significance of visualisation as well as its use in different contexts. The planned approach and requirements are also covered in this chapter.

2.1 Simulations

A computational simulation is an approximative reproduction of an event or behaviour of a given system based on a model. The model is described mathematically, often with a language that is tailored for this purpose, as a program, often called a model in this context. It is to be seen as a tool to examine a given system without a need to conduct physical experiments. All models used are simplified for the purpose of simulation - either because the designer lacks detailed knowledge about the system, or because of the system’s complicated nature. Even if the designer could describe the system in a precise way, simulation of the resulting model would be computationally infeasible[16]. Among the many needs for such abstraction is the fact that real numbers, omnipresent in physical models, can not be represented in a computer.

The simulator’s job is to calculate the possible output of a system in question using some computational (numerical or symbolic) methods. There are many ways to do that; the choice of the method basically depends on the model’s characteristics and type of the sought solution. Simulators may differ in many regards, including support for:

1. continuous or discrete notions of time,
2. declarative language features,
3. hierarchical/modular models,
4. variable dimension of simulation state space,
5. hardware-in-the-loop simulation,
6. distributed simulation,
7. visualisation of the simulation results,
8. checking of safety or liveness properties [23],
9. state space exploration through reachable state over-approximation [11, 7].

Features 7 and 9 are the focus of this project - 3D visualisation of the reachable set over-approximation produced by interval-based (over-approximative) simulation.
Interval-based simulation\(^1\) is a relatively new approach for dealing with the uncertainty rooted in the numerical methods used by simulators. Compared to simulation based on traditional numerical methods and floating-point numbers (henceforth simply *traditional simulation*), interval-based simulation uses structures based on intervals to represent numbers. It performs operations in a way that ensures that the result is a guaranteed over-approximation of the result that would be obtained if computations were performed using real numbers and continuous time. This is useful in several domains [10, 4, 18]. To understand how interval-based simulation works and what its benefits are, consider the following system, its model and the corresponding simulation result (shown in Figures 1 and 2):

```plaintext
model Main(simulator) =
  initially x=0, x'=5, x''=0,
  always if  x == 1 then x = 0 else x''=0,
  simulator.endTime+ = 1, simulator.timeStep+ = 0.001
```

The system that we will consider is a sawtooth generator, drawing a sawtooth on an oscilloscope's screen. The drawn line starts at 0 and grows linearly until it reaches 1. It is then reset to 0 and the cycle continues in this manner. Ideally, the display shall present a perfect sawtooth wave. However, when simulating this model with a traditional simulation the output can be different, as computers do not process continuous signals, and instead work in a discrete manner. Simulating this particular model in Acumen, using traditional simulation, the condition \( x == 1 \) will not be met as it is too strict. The variable \( x \) will, come very close to 1, but will not be exactly 1.

Why would the \( x \) not be exactly 1? Looking at the code above, the time step is 0.001; so increasing the value \( x \) by 0.001 1000 times should result in 1. The problem is that floating-point numbers are used to represent such values, and 0.001 is not exactly representable as an (IEEE754) floating-point number. Instead, at every step a small error is introduced, causing the check \( x == 1 \) to fail, resulting in an incorrect simulation result (Figure 1). A savvy user could of course modify the given condition to \( \text{if } x >= 1 \), but ought ideally not have to resort to such work-arounds. Instead, focus should be on precisely modelling the design.

The same model run with an interval-based simulation will result in intervals containing all possible outcomes of the simulation (Figure 2). What it means to the designer of a system is that while traditional simulation is erroneous in its foundation, interval-based simulation will consider all possible outcomes and return an uncertain, but rigorous, answer. This means that the user if always given a correct result with an explicit margin

\(^1\)In practice, more elaborate structures called *enclosures* [31] are used to represent numbers in simulators. Enclosures are based on intervals, but maintain additional information about the function that generates the number, that can be used to prevent error from accumulating.
The main obstacle to the adoption of interval-based simulations is that the accumulation of error generally leads to very wide, uninformative intervals.

2 The main obstacle to the adoption of interval-based simulations is that the accumulation of error generally leads to very wide, uninformative intervals.

---

Figure 1: Simulation of a sawtooth generator based on floating-point numbers; the execution has clearly gone wrong.

Figure 2: Interval-based simulation of a sawtooth generator; the simulator assumes that conditions are either fulfilled (creating a sawtooth wave) or not.

Understanding the mechanics of interval-based simulation is not essential to the outcome of this project; it is however important to understand what the results of interval-based simulations represent in order to map them to 3D.

- The result of an interval-based simulation with no uncertainty will initially be almost exact (small intervals are produced, that look similar to the simulation based on floating-point number) [Figure 3]. Different sources of error will, in general, inevitably lead to the accumulation of uncertainty.

Figure 3: For a model with no uncertainty, the interval-based simulation results in an output that looks like a line (left picture), but at a closer look one can see that small intervals occur (right picture).

- The result of an interval-based simulation with uncertainties will produce an interval (a span) of values due to over-approximation and processing multiple possibilities.
In parallel to the increasing complexity of the simulation, the demand on storage of output-values will grow and affect the difficulty of managing these values[7][Figure 4].

Figure 4: For a model with uncertainty, the resulting output will include clearly visible intervals.

2.2 Acumen

Acumen is an open-source simulator providing a modelling language and tools to conduct simulations using different methods (simulation based on floating-point numbers, intervals, etc.) [Figure 5].

Figure 5: Acumen, main window.

To conduct an investigation of a given system using Acumen, one would first need to describe a model of the system and its behaviour using the Acumen language, then run the simulation and lastly study the outcome in form of either raw values, plots or 3D visualisation (all depending on user’s preferences and chosen simulation method; it is accordingly not supported for interval-based simulation results).

Acumen supports models with both continuous and discrete dynamics (including combinations of these, which is referred to as hybrid systems), dynamic creation of objects
(e.g., one can model an object that will divide into smaller fragments on collision) and much more\[30\]. To support all that, Acumen’s source-code can be divided into different parts, including interpreters that process the Acumen language into representations closer to mathematical models and generate the simulation results, an integrated modelling environment, and 3D visualisation facilities based on an open-source Java engine\[15\]. Most of Acumen is implemented in Scala.

2.3 Scala

Scala is a relatively new language, released in 2003 by Martin Odersky. At first glance it resembles Java - "Scala is designed to interact well with mainstream platforms such as Java [...] It shares with [Java] most of the basic operators, data types, and control structures"\[22\]. Indeed, Scala is easy to intermix with Java and Java libraries, but it also provides enough functionality to be considered a complete language with its own place in modern software development.

Scala is a functional and object-oriented language, particularly appreciated for its multi-thread capabilities. It allows programming on a higher level of abstraction and is therefore considered an expressive language. Compared to lower-level languages, parallel and concurrent programming may be less error-prone in Scala, making it easier to scale up certain types of applications\[29\].

Scala’s functionality impacts Acumen on multiple fronts. This language’s similarity to Java does not stop at the syntax - Scala, just like Java, supports multi-platform execution of programs by leveraging a virtual machine. Scala supports interoperability with Java and using Java libraries, as in the case for Acumen’s 3D-mapping functions (using the 3D engine jPCT). In summary, further extensions of Acumen ought to be carried-out in Scala.

2.4 3D environment

The 3D environment in Acumen is based on jPCT - a light-weight engine implemented in Java, that supports most platforms (Mac, Windows, Linux and mobile operating systems). It is a self-sufficient library that provides a reasonably complete set of APIs and supports different types of 3D models and textures. Most importantly, jPCT supports software rendering, which is utilized by Acumen\[15\] to provide fully portable support for 3D visualisation.

The software rendering engine used by Acumen influences the 3D mapping that is currently implemented, as well as further extensions to this functionality. In particular, the amount of calculations needed to render a scene is high, which makes the lack of hardware acceleration capabilities noticeable. Moreover, previously performed simulations and their 3D-rendered visualisations are not saved and are thus not available for
future use, except for temporary viewing while Acumen is running. This can lead to situations when a big simulation has to be repeated and re-rendered in order to be examined, even though the model has not changed.

2.5 Visualisation

The importance of implementing a method for mapping of intervals into Acumen is endorsed by a study by Larkin and Simon[14] that points out the benefits of visualising raw data to improve comprehension:

- grouping related information minimizes the search process,
- strategic location of grouped information diminishes the need to match related data in memory,
- the human brain supports fast recognition of visual motives.

What Larkin and Simon are saying, is that visualisation of raw data can decrease the work a person has to put into the analysis of simulation results, which is beneficial to the understanding of the data[5].

Visualisation is especially important today, considering the increasing amount of computing power that is readily available for making vast amounts of data generated by computers understandable. A human being is only capable of dissecting a small amount of information represented in raw form, and that ability decreases relative to the complexity of the analysed information. To our aid come computer-generated graphics that make it possible to translate large numbers of variables representing different properties [Figure 6]. These translations are much simpler to comprehend as the part of the brain responsible for dealing with optical stimuli is equipped with a greatly developed skill of pattern recognition[8].

![Figure 6: Comparison of a table with raw values and their visualisation.](image)

Examine relationships between parameters is important when analysing results of simulations. Thus visualisation of interval-based simulation results ought to be supported by simulators, as it facilitates understanding and learning.
2.5.1 Visualising results of simulations of hybrid systems

There are different techniques for visualising results of simulations of hybrid systems that are worth mentioning:

- **Function plots.** These can be used in different ways, depending on the focus of the visualisation:
  - Plotting variables against time,
  - Plotting variables against variables,
  - Plotting variables against their derivatives.

- **2D diagrams.** These are used as formal models [13] and visual programming languages (for example in Simulink[27] and Modelica[19]). For some domains, such as 2D diagrams of circuits, these models can be highly declarative.

- **3D models.** These are widely used in conjunction with hybrid models in different domains. This project, for example, uses 3D models of the system in question and connects these with the mathematical model through state variables.

2.6 Related work

Elements of our method may be found in research conducted by related projects. This includes work on general hybrid systems verification tools, that perform computations with interval methods, as well as more domain-specific tools, that take into account uncertainty to improve the robustness of simulations.

**CheckMate and other hybrid systems reachability analysis tools** CheckMate is a “tool for modelling, prototyping, simulating specific situation and [...] verifying hybrid dynamic systems[... ]” [26]. It is related to our method as it supports visualisation of an over-approximation of the set of reachable states of a hybrid system; however, while we propose a method for visualising uncertainty in terms of a 3D model specified by the user, tools such as, CheckMate and SpaceEx[28] [11], visualise raw values of uncertain variables in 3D.

**CarSim** CarSim[6] is a simulator designed to accurately reflect behaviour of different vehicles, based on highly detailed models. This tool supports visualising uncertainties as families of 3D-objects; however, to our knowledge, CarSim does not support visualisation of interval-based simulation results.
Other work  Another example of existing work related to this project is the visualisation method proposed in Hybrid Automata as Modelling Approach in the Behavioural Sciences[3], where the authors apply visualisation of raw values and a 2D model of reality for simulations based on Monte Carlo methods. However, the overall approach is different in the sense that Monte Carlo methods produce many simulations of the same system, executed with randomized inputs, meaning that the simulations are not guaranteed.

Conclusion  To our knowledge, no available work could be used as a basis for this project. Existing approaches all solve problems that are different from ours, and none of them present a direct technique/method for visualising interval-based simulation results in a way that meets the requirements outlined in section 1.2.

That said, CarSim’s approach for visualising uncertainties as families of 3D objects was a starting point for our method. It clearly presents the intervals and 3D models in question. The resulting series of 3D objects are easy to comprehend and the only drawback is the lack of rigour in such a solution.
Figure 7: Comparison of a sparser series of objects (left picture) and a denser series of objects (right picture); if the number of objects in series was infinite, a hull conservatively representing the interval in question would be created.

3 Method

This section goes through the steps needed to fulfil the requirements stated above.

3.1 Approach

This section considers the criteria described in section 1.2, and outlines an approach for meeting them, taking into account that the outcome must be both rigorous and usable.

To visualise the uncertainty present in interval-based simulation results, the model must be mapped in a way that makes it easy for the user to understand that the viewed results are intervallic and clearly and conservatively present the relationships between the input parameters, as well as the magnitude of presented values.

To achieve that, manipulation of 3D objects according to the simulation output is necessary; a model that specifies an uncertain position for a 3D object will produce an elongated (in one or more dimensions) object that will be further affected by possible uncertainty in rotation, etc. The 3D representation of the object with uncertain position may be achieved by:

1. producing a number of copies of the object within the limits specified by the given enclosures, [Figure 7]

2. using orthographic projection of the object to produce a shadow zone with a duplicate of the object as the projection plane [Figure 8].

An additional solution is a hybrid between these two methods, where 1 is used to produce an under-approximation - making it easy to see what the object represents, and a rough version 2 would be used to produce a safe over-approximation. These are then presented at the same time to obtain the benefits of both methods.

In addition, elongation vectors must be gathered in order to achieve a correct model (see Milestones 1 & 2 - Acquiring relevant values by extraction).

The following tools were used to develop the method for mapping of uncertain values in Acumen:

- Eclipse with Scala IDE[24],
Figure 8: Orthogonal projection of an object against a duplicate of that object creates a hull that conservatively represents the interval in question.

- GIT[12],
- Acumen[1],
- SBT (build system)[25]

GIT is officially used in the Acumen project, so it is needed in order to download current version of Acumen source code. SBT (build system) is officially used in the Acumen project, and it is necessary in order to compile Acumen. Eclipse with Scala IDE has been recommended by Acumen developers; it comes with certain features that are useful - especially for developers new to Scala - such as Autocomplete and Debug.

3.2 Requirements, delimitations and results

This section gives an analysis of needs that this project has to fulfil in order to present a complete solution that satisfies Acumen’s users and developers.

3.2.1 Requirements analysis

To ensure appropriate visualisation of simulation results, an analysis of two potential Acumen user groups and their needs was conducted.

- The first group is the Users that will use the mapping method to analyse the results of interval-based simulations.
- The second group is the Developers that will use the mapping method to debug of the Acumen implementation.

Work focused on supporting two use cases of Users:

1. Robust simulation [10] - a simulation whose results are valid under perturbations of model parameters. For example, in such a simulation, initial conditions may be
uncertain (corresponding to intervals with non-zero widths), yielding a simulation of a range of scenarios. The rigorous nature of the simulation result may or may not be essential to the user.

2. Rigorous simulation [21] - a simulation whose results are guaranteed to be accurate, e.g. making them applicable to the analysis of safety-critical systems.

Analysing these two different users’ needs, as well as the above use cases, yields a number of features that must be implemented within the mapping method in order to make it usable and cover the different scenarios that can occur while investigating a system. The following cases must be supported:

- a model with absolutely no uncertainty, with a simple (linear) differential equation,
- a model with uncertain initial values,
- a model with uncertain parameters,
- a larger model with multiple modules.

To support already existing 3D capabilities and give the user an ability to model desired simulations/visualisations, some of Acumen’s 3D-related features must be considered:

- Supported 3D model types: Box, Sphere, Cylinder, Cone, Custom objects.
- Supported object manipulation parameters: center, size/radius/length, rotation.

3.2.2 Requirements fulfilment plan

A number of requirements has been agreed upon in collaboration with one of Acumen’s developers. To obtain a proper structure while implementing a method for 3D-mapping, the development process has been divided into two milestones, focusing on various aspects of the requirements. The first milestone is a simple, functional solution to the problem. The second milestone is a more elaborate solution covering all requirements.

- Both milestones
  - Box, sphere, cylinder, cone and custom objects are to be supported,
  - Center, size/radius/length and rotation parameters are to be supported.
- Milestone 1
  Implement use case 1 by visualising the range of scenarios that corresponds to uncertainties in the simulation results.
Extreme-values are to be represented in visualisation,
The shortest path between the extreme-values is to be covered by duplicates of the given model, indicating the unity of this group of models,
Uncertainty in every parameter and every possible combination of these parameters are to be represented in the visualisation.

- Milestone 2
  Implement use case 2 by visualising a (safe) over-approximation of the rigorous simulation result.

3.3 Delimitations

The intervals represented in the visualisation must include the correct values; however, the representation may also include additional values, meaning that it can be an over-approximation.

3.3.1 Validation of results

To validate results of this project, the method developed for mapping uncertain values to 3D models in Acumen is to be tested manually using different test cases and models that cover the area described in Analysis of needs. A final test with a thorough coverage of test cases, as well as an acceptance test are success criteria for this project.

3.3.2 Testing

A test specification has been designed (see Appendix 1) to ensure that all requirements are met, and so that third part stakeholders have a possibility to control that the project has been successful. The test specification goes through 6 different models that are based on the requirements stated in this project. Criteria for assessing all test results are proposed.

Such tests are a good way to rate the outcome of a project, but can also be used in educational purposes. Each model supplied in the test specification has been designed to tackle different problems that a user can meet in his work; a use for each model is suggested.

3.4 Milestones 1 & 2 - Acquiring relevant values by extraction

Regardless of the technique used to elongate objects, the values that are related to the positioning and elongation of the model must be acquired. Acumen provides its own type of a list called GStore that contains all the simulation data. That data is extracted and parsed, so that certain information (for example the delta - the difference between
the upper and lower bounds - of the position-values) becomes available for use within the 3D-environment. Depending on the number of objects in the simulation, GStore will have different characteristics with regard to the number of objects contained within it; the extraction-method (addTo3DStore) will make them available in the same manner in a list called _3DStore.

However, addTo3DStore does not recognize some values that are needed to process interval-based data. Therefore, it has to be altered so that following information is made available in _3DStore:

- minimum-value of the position and the delta of its limits,
- minimum-value of the rotation and the delta of its limits,
- maximum-value of the size,
- minimum-value of the colour.

These values can subsequently be imported and processed while creating the visualisation.

3.5 Milestone 1 - Visualising intervals in an accessible way

The technique used in the initial implementation of the mapping-method is simple and not rigorous. It is, however, a functional approach to the problem.

The method is employed if the simulation in question is performed using the Acumen Enclosure semantics - so if the simulation is interval-based. It starts by checking the delta value for the position of the given object in all three dimensions and divides that value in relation to the size of the object; that way the number of objects needed to cover the shortest path between extreme values is established. Depending on the result:

- if the delta equals zero, draw one object in the unique position specified by the minimum value,
- if the delta is not equal to zero, draw two objects in the extreme positions, and then proceed to fill the space between them with the object, overlapping each other to ensure that the shortest path between the limit values is covered.

The same treatment is applied to the rotation of the object. To provide a proper animation, it is crucial to remember to time-stamp the objects in relation to the given frame. It is also favourable to choose a rather small intersection-constant for the overlapping of objects (when dividing the delta with the size of the object); otherwise the number of models drawn between the limits will become too high to render them efficiently.
3.5.1 Assessment of the method

The approach described is not rigorous, as parts of the true elongated object will not be covered [Figure 9]; in general, the representation is only correct when the number of drawn objects is infinite, clearly not a feasible approach on a computer. This means that the visualisation will generally be incorrect in some points. The described method does however mediate supposed information in a useful manner, as long as the user is aware of what the visualisation represents. This method is inspired by CarSim’s approach to visualising interval values (see Related Work). While not rigorous, and thus not sound with respect to the requirements, this approach is functional and has a potential for further development. Moreover, this approach is relatively uncomplicated and therefore would be easy for Acumen developers to debug and maintain in the future, while fulfilling all requirements but one.

![Figure 9: A series of two spheres is created to illustrate one object’s possible positions. The green areas are the places of error that should be covered with solid model.](image)

3.5.2 Motivation

As stated in the "2.5 Visualisation", studies show that grouping related information minimizes the search process. Visualising intervals by constructing a series, "a family", of objects is essentially grouping related information. Advantages of such a practice are thus supported by psychological studies - it minimizes the search process, so the user saves time and the amount of work that he normally would have to spend in order to understand the results of his simulations.

Another reason for using this method is the fact that Acumen uses jPCT (see 2.4 3D environment)- 3D engine that offers somewhat limited possibilities for 3D-manipulation. Those limitations are the price one has to pay for the engine’s portability. The engine does not require support for hardware acceleration and can thus be executed anywhere where Java is supported. The lack of support for projection, and many other desirable methods, the palette of tools that jPCT supports does not give 3D-developers much to work with. The method chosen utilizes available options, so there is no need to alter the engine - something that would have been a very time consuming path to take.
Finally, the chosen method does not differ much from the 3D visualisation of traditional simulations that is already implemented in Acumen. The 3D models are the same, as well as the camera navigation in the 3D-environment. This means two things: users that are already familiar with Acumen’s 3D environment will feel familiar with the 3D visualisation of uncertain values, and developers will be able to quickly understand the source code, which gives them opportunity to easily debug and extend the method.

3.6 Milestone 2 - Visualising intervals in a rigorous way

The main concern of this milestone is the rigour of visualised data. The visualisation has to indicate all the points in space where the visualised object could possibly exist.

This can be achieved, without losing the good appreciation of the model’s state provided the approach from Milestone 1, by a simple extension of that approach. A semi-translucent block containing the series of object copies (from Milestone 1) is added, and it’s size is be modulated so as to enclose the points in space that are a possible result of the simulation (Figure 10). To do this, the relevant data is gathered - length, height and width of the model used. This information is then used to calculate the space diagonal of the object in question. Crucially, all objects must be interpreted as polyhedra.

A space diagonal is the longest straight line in a polyhedron (Figure 11:I, d). It can therefore be used to ensure that the over-approximation of an uncertain value bound to a 3D model is big enough to cover all possibilities. The calculation of a space diagonal requires the knowledge of sizes in all three directions (Figure 11:I, a,b,c) and the use of the Pythagorean theorem. Following Figure 11, we may obtain d by observing that 

\[ d^2 = c^2 + e^2 \]  

(Figure 11:II) where 

\[ e^2 = a^2 + b^2 \]  

(Figure 11:III). The resulting equation is 

\[ d = \sqrt{a^2 + b^2 + c^2} \]

Then, d can be applied to the block that encloses the uncertain representation of a given model (see Milestone 1) to alter its size accordingly and ensure that the over-approximation is of a proper size.

These calculated values, together with the sizes of the intervals, make for a block that can be put on the uncertain representation of a model (see Milestone 1) visualising the over-approximated values in question. Using setGlass, provided by jPCT, that block becomes semi-transparent and thus allows the user to see through it and to observe the duplicated objects inside the block.

Figure 10: An illustration of the second milestone’s approach towards a rigorous solution of visualising enclosures.
3.6.1 Assessment of the method

This method creates a rigorous visualisation of the model in question, and therefore fulfils the requirements specified in section 1.2. On the other hand, the result will be over-approximated, sometimes highly. The precision (information content) of the resulting 3D visualisation depends on the model in question and the size of intervals produced by the enclosure semantics. Most importantly, the user can rest assured that the visualisation envelops the right values.

This method alone fulfils all the stated requirements. The result is rigorous and thus safe to use in place of Acumen’s existing data visualisation capabilities (plots and tables). Moreover, this method is a good supplement to the Milestone 1, and when put together, the resulting solution becomes not only mathematically correct but also easy to understand.

3.6.2 Motivation

As stated in "2.1 Simulations", a key point of using interval-based simulations is the need for a rigorous answer. While other simulation methods will provide a result that is erroneous in its foundation, interval-based simulations will give an uncertain, but rigorous answer. It is therefore critical that the visualisation maintains this property, enclosing the correct answer, even if a margin of error will be included in the representation. Using space diagonal to create a 3D object enclosure, given that the representation can be over-approximated, is mathematically correct. It gives the user an assurance that no matter what system, model, 3D-mesh and/or parameter he works with - its representation will be rigorous.

Moreover, as stated in "2.5 Visualisation", the human brain supports fast recognition of visual motives. A 3D block is one of the simplest geometrical figures that could be used; no unnecessarily complex features are used. Its simple geometry is easy to grasp - the sides of the block are the limits within which the correct result can be found. For example, the user can easily see whether or not two semi-transparent blocks intersect.
4 Results

This section considers the results of this project and includes an analysis of the performance of the prototype implementation.

4.1 Visualisation

The first method that was developed for Milestone 1 (see Figure 12a), creates a series of objects along the vector/vectors of elongation created by the fact that the visualised simulation contains uncertain values. This series of objects, when put together, create an illusion of an elongated object, and thus mediate the interval of possible outcomes provided by the simulator. The result is not rigorous, and thus strictly put incorrect, but the error is explicitly represented.

The second method (see Figure 12b) creates a semi-transparent block that is applied over the series of objects from Milestone 1. That block is an over-approximation of values already represented in Milestone 1, and this addition to the earlier solution makes the final result rigorous.

Using these two methods together makes for a combination of both accuracy and useful mediation of the interval provided by the simulator; the series of objects show what the source-object is, and together with the semi-transparent block, clearly illustrate the interval of possible outcomes of the simulation.

4.2 Source code

The prototype source code produced has been reviewed and adjusted to the convention used in the Acumen project. Every piece of code that is to be integrated with Acumen’s public code base is reviewed by people administrating the Acumen project. In the same way as every good product must go through quality control, so does good software; the administration of the Acumen project must ensure that their software, when released to the users and other developers, lives up to certain standards set by their organisation.
The integration of the outcome of this project with the public code base of Acumen is therefore a measure of its quality.

4.3 Run-time performance

Manual tests show that all of the requirements stated in this paper are met by our solution. Moreover, understanding the interval based simulation results has become less time consuming, which is especially useful in complex simulations, where the conventional raw data/plot based analysis can be inefficient.

Figure 13: A chart of the run-time for Milestone 2 (”With block”), Milestone 1 (”Without block”) and with no visualisation at all (”No visualisation”) using models1 with different complexity2.

The performance of the software itself is altered due to the rendering needed in the 3D environment. The relation between the number of 3D objects and the run-time is clear; the processing of big models with many parameters requires more computational resources. It is worth mentioning, however, that Acumen makes use of multi-threading which compensates the performance degradation when multiple cores are available. The run-time chart (Figure 13) shows that the time needed to process and render models using the method described as Milestone 2 (blue line, ”With block”) and Milestone 1 (red line, ”Without block”) increases significantly with the number of 3D objects. For

---

1Used in experiment: Intel Core i7-3610QM, 6GB RAM, Samsung 840 Basic; running Oracle Java 6 on Windows 7.
2Experiment source[20].
simpler models, both methods are just slightly more demanding than the reference - a simulation with no 3D visualisation at all (yellow line, "No visualisation").

4.4 Test results

The test results \(\text{(see Appendix 1)}\) are positive. All of the tests passed, and users can for the first time use our 3D visualisation as a complement to the tables of raw values and 2D-plots that are presently available in Acumen.

The tests show that all of the requirements that we aimed to meet are indeed supported by our implementation. Moreover, during testing, we ensured that no other functionality in Acumen has been altered; everything works as it did before our extensions (with the addition of the visualisation).

Beside the tests shown in \(\text{Appendix 1}\), continuous testing has been conducted during the development process. The automated test suite that is part of the Acumen code base, as well as numerous manual tests, which were not officially documented, give us an additional assurance that everything works as it should.

One could argue that automated testing of our part of the implementation should be conducted; however, it is important to remember that the main goal of visualising is to give the user an easy way to read information. It is therefore, in our opinion, much more revealing to test everything manually - from the user’s point of view. Automated tests of our predominantly graphical functionality would also incur a maintenance effort from the Acumen developers. This effort may be unacceptable over time, as tests of such code can be be prohibitively brittle, especially in the context of a research-oriented code base.
5 Discussion

All of the goals set forth at the beginning of this project have been met by our proposed solution. Users can now run an interval-based simulation and review the results as a 3D visualisation. This visualisation presents all the intervals rigorously (see Test specification, Appendix 1). As a result, the accessibility of these simulation results has been improved significantly.

As discussed in section 2.6, there is no other software visualising interval-based simulation results, thus no comparison can be made. It is, however, important to point out that not having any convention to follow and not having any solution to mimic, has been a good exercise in creativity and allowed new, fresh solutions to be tested.

5.1 Flaws

The major flaw that we have observed is the significant increase of the run-time of a simulation, caused by a sometimes massive number of 3D-objects, rendered simultaneously. Acumen provides the user with a measurement of the run-time for every simulation made which has been used to create the Run-time chart in Figure 13. Measures were taken to ensure that our measurements would not be affected by concurrently executing processes on the test machine, and the error margin was below 1 second.

5.2 Strengths

Even though the runtime of simulations has increased, our method makes the results more accessible and easy to understand. In the long run, we argue, the user actually saves time, as he now can investigate the results of his simulations much more conveniently than before. Examining raw data/plots is time consuming, even more so, when the results are intervals. Thus, our method contributes to making rigorous simulation methods applicable to the analysis of larger (more realistic) models.

Moreover, we argue that our visualisation method makes interval-based simulation more approachable. It can now be understood by laymen, which is important for spreading the knowledge about this novel and under-appreciated simulation technique. Self-explanatory figures can now be used in documentation/presentations dealing with interval-based simulation, making the whole experience much more tolerable.

5.3 Why fight for visualisation?

We want to take the opportunity and emphasize the importance of visualisation in modern computer science and, with that, emphasize the importance of this project. Young people, not old enough to remember searching for information in encyclopaedias,
cannot really understand the meaning of the ”age of information” that we recently have stepped into. Those that do, or think they do, really do not. The fact is, nobody does, because the amount of information that we can access, or produce, with help of our computers is unimaginable. People can imagine one litre of water, or one kilogram of sand - but information is somehow an abstract thing - one cannot really imagine the sheer amount of information that is at our fingertips.

All this information, though, is virtually useless if we cannot read, interpret, and understand it. Preferably, we would like to understand it and see the big picture instantly. Unfortunately, it takes time and practice (and quite an amount of expertise) to learn to operate with some types of information; just as reading the algebraic notation used in chess will be a piece of cake for a Grand Master, an amateur player will have big trouble going through moves like Qe5xg6+.

To our aid comes visualisation. Representing chess notation with a picture of the chess-board, together with all the pieces on it, will be instantly understood by everyone (assuming they know chess). In the same way, every type of information will be much more accessible if represented with figures, pictures, colors, 3D models, etc. It all comes down to the fact, that human being’s skill of optical pattern recognition is absolutely remarkable (see 2.5 Visualisation).

Now, modern computer science gives us possibilities that we have never dreamt of. A regular personal computer can produce incredible amounts of information, and methodologies become more and more complex, resulting in correspondingly complex results. But if this evolution of computer science, if all these possibilities are to make sense - we must start supporting their visualisation. It does not matter that we can produce a table consisting of 30000 raw values in a matter of seconds, if investigating those values will take 3 weeks; no one would ever try and decode such an amount of information, and the whole thing becomes merely an exercise in computation. Showing those 30000 cells of raw values, directly, as the bouncing ball (or anything else) that they represent, should be expected in this age of information.

As for the visualisation of uncertain values, produced by interval-based simulations in Acumen, it is very clear that this project is important. Traditional simulations are very good for most applications, but the truth is that such simulations will never be able to capture the true nature of the modelled systems. This is not only because of the numerical methods or the computer implementations that necessarily approximate values, but also due to other factors. For example, we can never measure the position of particles exactly. As discovered by Landsberger and formulated in the principle called the observer effect, when we measure the position of a particle, we necessarily also alter it. Dealing with these fundamental properties of reality directly in simulation tools (that are, in the end, designed to emulate it), by representing and
propagating uncertainty from the model to the simulation result, therefore seems like a good idea.

With computers taking on an increasing share of the work in the process of designing cars, aeroplanes, space shuttles, controllers, pacemakers and much more - we are in need of reliable simulations; simply because, in some cases, lives are at stake [9]. Introducing visualisation of uncertain values makes it easy to understand interval-based simulations; it can therefore encourage system designers to use them, improve them and invest in them. The more interest interval-based simulations can gather, the faster they will develop, and our method makes it accessible to a broader public. In summary, we feel that this project makes a small contribution to a great project that in coming years could emerge as an invaluable tool in the cyber-physical system designer’s tool box.
6 Conclusions

The method presented in this paper allows visualising results of interval-based simulations in Acumen. It has been implemented in the simulator, Acumen, and fulfils all of the set requirements. It supports all of the models and parameters that were required, it mediates the interval nature of values, and it does so rigorously.

Acumen lacked the possibility of visualising interval-based simulation results in 3D, and to our knowledge, there is no other software that provides that feature, which makes this project timely.

This project has been a great opportunity to learn Scala and deepen our knowledge of high-level programming. Our understanding of simulations and simulators has grown significantly, especially in the terms of interval-based simulations, which are a relatively new concept. This has been the first time we got a chance to work with such a well-developed software, extensive not only in code, but also in structure.

6.1 Further work

From the perspective that this project gave us, we suggest two lines of further work:

- Orthogonal projection - a method of visualisation of interval values using orthogonal projection that would create a perfect convex hull that would be rigorous and that would mediate the meaning of the model in question. An issue we foresee here is how to handle rotations. Our suggestion is to use a rotation matrix, but using some other technique is possible; the important part is to ameliorate of the main limitation of our solution (see Delimitations).

- Mixing visualisations of interval-based simulations with those of traditional, traditional simulations. When simulating larger models and using our solution to visualise them, e.g. a car driving on a road, all parts of these models will be represented as intervals. It is however obvious that in some cases it is not only unnecessary, but also disturbing to have all of the 3D objects in question rigorously represented in the visualisation. We suggest that it would be a major improvement if the user could choose which of the objects are to be represented in a traditional manner, and which are to be represented in the way we propose in this report.
References


[20] **Models.** URL: www.dropbox.com/s/dj0vk8yuscpl2mh/Models.rar?dl=0.


[27] **Simulink.** URL: www.mathworks.com/products/simulink/.


Appendix 1 - Test specification

Visualizing uncertain values

- Sphere,
- Box,
- Cylinder,
- Cone,
- Custom meshes (objects) of your own choice

Table 1. List of models to be tested.

Table 2. Criteria when judging results.

1. Are the extreme values represented in the visualization? yes/no
2. Is the shortest path between the extreme values covered in objects? yes/no
3. Do any new errors appear in the console? yes/no
4. Does the over-approximating, semi-transparent block envelop the series of 3D-objects within it? yes/no
5. Compare the 3D visualization with 2D plots. Can you draw a parallel between these two? Do you arrive at the same conclusions about the model in both cases? yes/no
6. Repeat the test; does the visualization differ - in any way - from the previous one? yes/no
1. A model with no uncertainty, with a linear differential equation.

```plaintext
model Main(simulator) =
  initially
  x=5, x'=0, x''=-10,
  _3DView=(),
  _3D = ()
  always
  x' = -5,
  simulator.endTime+ = 1,
  simulator.timeStep+ = 0.001,
  _3D = (Sphere size=0.2 center = (0.0,0.0,x) color = green)
```

Model 1

a. Simulate Model 1 with 2015 Enclosure semantics in Acumen.
b. Check the console for possible errors.
c. Visually analyse the visualisation as specified by Table 2.
d. Change current model (here Sphere) to the next model in the Table 1.
e. Repeat the test (go to a.) until all models are tested.

**Result:** example screenshot, Model 1 using custom object (car)

What is tested:

- that a model with no uncertainty is represented correctly;
- that all required 3D-objects are supported and can be used;
- that the representation is rigorous.

**Errors:** none (see screenshot)
Test passed: yes

When to use?: this type of model can be used instead of classic, floating-point number based simulations, when the user doesn’t need to include uncertainties in his model, but wants to be sure that the overapproximation in numerical methods will be accounted for.
2. A model with uncertain initial values.

```plaintext
model Main(simulator) =
    initially
    x=[4 .. 5], x'=[0 ..1], x''=[-10 ..-20],
    _3DView=(),
    _3D = ()
    always
    x' = -5,
    simulator.endTime+ = 1,
    simulator.timeStep+ = 0.001,
    _3D = (Sphere size=0.2 center = (0.0,0.0,x) color = green)
```

**Model 2**

b. Check the console for possible errors.
c. Visually analyse the visualisation for unwanted behaviour.
d. Change current model (here *Sphere*) to the next model in the Table 1.
e. Repeat the test (go to a.) until all models are tested.

**Result:** example screenshot, Model 2 using sphere

What is tested:
- that a model with uncertain initial values is represented correctly;
- that all required 3D-objects are supported and can be used;
- that the representation is rigorous.

**Errors:** none (see screenshot)

**Test passed:** yes
**When to use?:** this type of model can be used when we want to simulate something where we either are unsure about some initial values, or where we simply want to see system’s behaviour for multiple initial possibilities with just one simulation.
3. A model with uncertain parameters.

```plaintext
model Main(simulator) =
    initially
    x = 5, x' = 0, x'' = -10,
    _3DView = (),
    _3D = ()
    always
    x' = -[4.0 .. 5.0],
    simulator.endTime+ = 1,
    simulator.timeStep+ = 0.001,
    _3D = (Sphere size=0.2 center = (0.0,0.0,x) color = green)
```

**Model 3**

a. Simulate *Model 3* with 2015 Enclosure semantics in Acumen.
b. Check the console for possible errors.
c. Visually analyse the visualisation for unwanted behaviour.
d. Change current model (here Sphere) to the next model in the Table 1.
e. Repeat the test (go to a.) until all models are tested.

**Result:** example screenshot, Model 3 using box

What is tested:
- that a model with uncertain parameters is represented correctly;
- that all required 3D-objects are supported and can be used;
- that the representation is rigorous.

**Errors:** none (see screenshot)

**Test passed:** yes
**When to use?:** this type of model can be used when we want to simulate something where we either are unsure about the system’s behaviour, or where we simply want to see simulate a system with multiple behavioural possibilities with just one simulation.
4. A larger model comprising multiple modules.

```plaintext
model Main(simulator) =
    initially
    x=5, x'=0, x''=-10,
    y=[4 .. 4.5], y'=[0 .. 1], y''=[-10 .. -20],
    _3DView=(),
    _3D = ()
always
    x' = -[4 .. 5],
    y' = -5,
    simulator.endTime+ = 1,
    simulator.timeStep+ = 0.001,
    _3D =
    (Sphere size=0.2 center = (0.0,-1,y) color = green,
     Sphere size=0.2 center = (0.0,1,x) color = red)
```

Model 4

a. Simulate Model 4 with 2015 Enclosure semantics in Acumen.
b. Check the console for possible errors.
c. Visually analyse the visualisation for unwanted behaviour.
d. Change current model (here Sphere) to the next model in the Table 1.
e. Repeat the test (go to a.) until all models are tested.

Result: example screenshot, Model 4 using spheres
What is tested:

- that a larger model comprising multiple modules is represented correctly;
- that all required 3D-objects are supported and can be used;
- that the representation is rigorous.

Errors: none (see screenshot)

Test passed: yes

When to use?: this type of model can be used when we want to simulate larger models that include a number of objects whose possible interaction between each other is of our interest.
5. A model testing parameters that are to be supported (size, rotation, center)

```plaintext
model Main(simulator) =
  initially
  x=5, x'=0, x''=-10,
  _3D = ()
  always
  if x <= 0 && x' < 0
  then x' = -[0.9..0.95] * x'
  else x''=-10,
  simulator.endTime+ = 3,
  simulator.timeStep+ = 0.001,
  _3D = (Obj center = (0,x,0)
       size = 1
       content = "car"
       rotation = (x,x,x))
```

**Model 5**

a. Simulate Model 5 with 2015 Enclosure semantics in Acumen.
b. Check the console for possible errors.
c. Visually analyse the visualisation for unwanted behaviour.

**Result:** example screenshot, Model 5 using custom object (car)
What is tested:
  ● that all parameters that are to be supported are represented correctly;
  ● that the representation is rigorous.
Errors: none (see screenshot)
Test passed: yes
When to use?: this type of model can be used when we want to simulate a model where uncertainty is either in size, rotation, etc.
6. A model with uncertainty in every possible parameter (combined)

```plaintext
model Main(simulator) =
    initially
    x=5, x'=0, x''=-10,
    _3D = ()
    always
    if  x <= 0 && x' < 0
    then  x' += -[0.9..0.95] * x'
    else x''=-10,
    simulator.endTime+ = 3,
    simulator.timeStep+ = 0.001,
    _3D = (Obj center = (x,x,x)
        size = 1
        content = "car"
        rotation = (x,x,x))
```

**Model 6**


b. Check the console for possible errors.

c. Visually analyse the visualisation for unwanted behaviour.

**Result:** example screenshot, Model 6 using custom object (car)
What is tested:

- that uncertainty combined in all possible parameters that are to be supported is represented correctly;
- that the representation is rigorous.

Errors: none (see screenshot)

Test passed: yes

When to use?: this type of model can be used when we want to simulate a model that combines several uncertainties at the same time.
### PLANERING

**Projekt:** 3D-Acumen – Visualizing Uncertainty  
**Projektgrupp:** 3  
**Handledare:** Adam Duracz  
**Datum:** 30/01/2015  
**Kurs:** Examensarbete  
**Utfärdare:** AP, HA  
**סאנד:** Amadeusz Pawlik

<table>
<thead>
<tr>
<th>AKTIVITETER</th>
<th>TID (när)</th>
<th>VEM</th>
<th>TIDPLAN (när), veckonummer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr Beskrivning</td>
<td>timmar</td>
<td>Initialer</td>
<td>5</td>
</tr>
<tr>
<td>1 Planering</td>
<td>55</td>
<td>19 11 10 8 3 3 1</td>
<td>55</td>
</tr>
<tr>
<td>2 Dokumentation</td>
<td>250</td>
<td>20 10 10 10 10 10 15 15 15 15 15 15 15 10 15 20 20</td>
<td></td>
</tr>
<tr>
<td>3 Studier</td>
<td>85</td>
<td>22 13 10 15 15 10</td>
<td>85</td>
</tr>
<tr>
<td>4 Programmering</td>
<td>230</td>
<td>10 15 16 15 18 18 18 19 28 19 18 11 5 5 15</td>
<td>230</td>
</tr>
<tr>
<td>5 Testning av simulering</td>
<td>85</td>
<td>10 10 10 10 10 10 15</td>
<td>85</td>
</tr>
<tr>
<td>6 Redovisning</td>
<td>30</td>
<td>10 10 10</td>
<td>30</td>
</tr>
<tr>
<td>7 Projektmöten</td>
<td>25</td>
<td>1 2 2 2 1 2 1 2 1 2 1 2 1 2 1</td>
<td>25</td>
</tr>
<tr>
<td>8 Reservtid</td>
<td>40</td>
<td>8 18 14</td>
<td>0</td>
</tr>
<tr>
<td>9 Planeringen är färdig</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10 Programmeringen är färdig</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Enkel 3d-presentation är klar</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Testningen är färdig</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Sluttest</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Summa antal timmar:** 800