Gabriel Rădulescu

Advanced Modelling and Simulation Techniques

A Stewart Robinson's opinion from his book,
"The Practice of Model Development and Use"

Ploiești, 2016

CONTENT

Foreword

This course on Advanced Modelling and Simulation Techniques is part of the master programme *Advanced Automation* (Faculty of Mechanical and Electrical Engineering at the Petroleum-Gas University of Ploiesti, Romania). Its aim is to provide the student with a clear understanding of the basic requirements for the model-based simulations successful implementation and use.

Although some examples are taken from the technical area, the course may generously cover almost all fields of activity where modelling and simulation are used as research and development tools. It does not focus on particular mathematical models, numerical methods or some specific software environments. Instead, the course emphasizes the importance of efficiently managing a project with two partners: the modelling/simulation specialist and the client organization.

The course closely follows the author's expertise in this field, as well as other ideas, opinions and approaches in the open scientific literature. Especially, it makes use of a valuable book, *The Practice of Model Development and Use*, written in 2004 by Stewart Robinson (University of Warwick, UK) – a renowned specialist in modelling, simulation and research management. The large citations indicated in this text were preserved in their form in order not to distort the original meaning, but they are blended with new/original opinions and examples, aiming to offer the reader an accessible and comfortable lecture.

The author

1.
Modelling & Simulation – an Overview

1.1. A Short Introduction

When the executive staff of a power plant estimates the facilities that are required in a future new power station, important decisions need to be made. Among other things, they have to take into account the future clients pattern (industrial or home individuals for instance), the number of power generators, the appropriate fuel type, the amount of safety devices and the number of leaving high-power lines. Also, the number of operating staff to employ and the required shifts they should work need to be determined. As the total investment is important, it is critical that these decisions are made correctly, while the management has to find an answer on how to determine the number of resources that are required in each area of the power plant.

One approach would be to finalize the investment, hoping that it works – but this seems to be a very risky option. Slightly better would be to rely upon some past experience with designing and managing power plants. A few calculations may help, but these are not able to cover the full complexity of the situation.

A more effective approach seems to be a simulation of the proposed plant. This could imitate the different clients' power needs and consuming behavior (depending on their pattern, as described above), the external flow of materials (fuel for generators, spare parts for maintenance and so on), other influencing (or disturbing) factors for the power generators and would act as a basis for planning plant facilities.

Indeed, simulation models are used by many organizations to plan future facilities and also to improve current ones. For instance, manufacturing companies simulate their production facilities; financial organizations simulate their assistance centers, while transport companies simulate their delivery networks. Of course, many other examples of simulation being used in practice can be identified.

This introductory chapter tries to answer three questions concerning modelling and simulation:

- What is a simulation?
- Why would an organization choose to develop and use a simulation model?
- When is simulation appropriate?

1.2. Defining Modelling and Simulation

Simulation models are in everyday use and so simulation is a concept that is ordinary to us. For example, when weather forecast TV presenters show us computer simulations of the weather system, we watch to the movement of a rainy clouds front for the next hours, days and weeks. The game consoles may also be mentioned here, as they simulate real activities like testing our speed drivers, adventurers or detective skills. But simulations not really need to be computer based: model railways and boats are typical examples of physical simulations. So, in its most general sense, the *simulation* term can be defined as *an imitation of a system* (Robinson, 2004).

Imitation implies copying something else. For instance, we may imitate different entities, from the work of a great artist (by forgeries) to the famous buildings (with smaller replicas). Also, computer aided design (CAD) systems providing imitations of production facility designs are an imitation of a business organization. All of these can be referred as a simulation in its most general sense.

But there is a key difference between these imitations and those examples described earlier in this section, which involve the passage of time (clouds migration on the sky, race car movement on its track and so on). The second set of examples does not imply the time as characterizing parameter. At this point, the difference between the *static simulation* (reproducing a system at a particular moment in time), and the *dynamic simulation* (imitating a system as it progresses through time) have to be emphasized (Law and Kelton, 2000).

As the term *simulation* is simply used instead of *dynamic simulation*, it is to be mentioned that this course is concerned only with dynamic approach(es) when imitating a (technical) system, while the focus will be on computer based simulations rather than physical simulations. Taking into account these considerations, the previous definition can be updated as follows: *simulation means an imitation (on a computer) of a system as it progresses through time.*

It may be useful to explain the concept of a system. In general terms, a system represents a collection of inter-connected parts organized for some purpose (Coyle, 1996). The weather system, for instance, is a collection of parts, including the sun, water and land, “designed” for the purpose of maintaining life. The above power plant is a collection of generators, additional equipments and operating personnel, with the purpose of supplying the energy needed by its clients. Many other examples may be identified, and the systemic approach is proven to be a very robust way of investigating the real environment.

Checkland (1981) identifies four main classes of system:

- *Natural systems*, whose origins lie in the origins of the universe (the atom, the climatic system, the natural hydro-systems).

- *Designed physical systems* that are a result of human design (a block of flats, an automobile, a factory).
- *Designed abstract systems*, that are also a result of human design, but do not have a physical consistency (mathematics, philosophy, literature).
- *Human activity systems*, consciously or unconsciously ordered (a family, a political system, an economic system).

All systems above can be simulated. This course, however, mostly focuses on designed physical and human activity systems, which best and completely describe the technical field of interest. For instance, a simulation developed for an automated refinery plant (a designed physical system) has to be completed with a simulation of its operating personnel behavior (a human activity system). In this situation the complex system cannot be regarded simply as either a designed physical system or a human activity system, but rather an interface between the two. As consequence, many of the situations in which simulation is used also lie at the interface between designed physical systems and human activity systems (Robinson, 2004). In the open literature, such cases are referred to as operations systems or operating systems (Wild 2002).

A second aspect of the last definition is the purpose of simulation models. Pidd (2003) identifies the purpose of models as *understanding*, *changing* and *controlling* the reality. In this respect, simulation leads to a better understanding of and/or identifying improvements to a system, becoming a primary support for the decision-making process.

The same author puts emphasis on *simplification*, as one of the most important features of the simulation models. Obviously, a hypothetic full-scale simulator for a given system, representing all system's details, is barely usable due to its complexity. On the other hand, even if it were possible, such a "complete" modelling approach is not feasible, since the time required to collect data on and model every aspect of a system would be excessive (Robinson, 2004).

Another aspect to be considered is the nature of simulation model use. A simulation simply describes the behavior of a particular system in the context of specific inputs. For instance, in the context of a simulated power plant, it might predict the minimum number of active generators during high-consuming periods. Of course, by varying the model inputs (specific clients' behavior and their particular power needs) and repeatedly running the model, one can observe their influence in the simulated system response. In this case, simulation is an experimental approach to modelling – so becoming a powerful analysis tool – since the user select an operational scenario and the model predicts the outcome. By continuous exploration, scenario by scenario, a better understanding of the real system can be obtained. Since new knowledge may lead to important improvements of the system, simulation should be regarded as a form of decision support system.

By adding these new aspects to previous definitions, simulation can finally be seen as *experimentation with a simplified computer-imitation of an operational system as it evolutes through time, for the purpose of better understanding and/or improving that system* (Robinson, 2004).

1.3. Simulation vs. other Investigation Techniques

This paragraph adopts three perspectives, in order to reveal the simulation necessity, its advantages and disadvantages when taking into account other tools and approaches for systems analysis and control.

1.3.1. Simulation in the context of operational systems nature

Usually, operational systems are subject to *variability*. This might be equally predictable (for example, changing the number of active power generators in order to meet the plant required output power) or unpredictable (such as the plant equipment breakdown). Both forms of variability are present in most operations systems.

Usually, operational systems are in fact individual elements of a global system, and so they are also interconnected, affecting one another. A change in one part leads to a change in another part of the system. For instance, if an active power generator is set up, additional operating personnel is also required. But it is difficult to predict the effects of the interconnections in a system, especially when variability is present (Robinson, 2004). Let us take the related example of a power plant. If considered totally free of any defective equipment during its normal operating life, the plant's needs with respect to personnel are strictly related to the number of active generators. But, as the reality is different, any unexpected and unpleasant technical event has to be solved by re-allocating technicians and ordinary workers. In this context it is possible that, at a particular moment, one generator will not be served by all the standard personnel and if something goes wrong additional problems appear on that power plant.

Although we describe both the system and its behavior as being *complex*, it is difficult to provide an exact definition of the word *complexity*. For our purposes it is important to distinguish between *combinatorial complexity* and *dynamic complexity*. Robinson (2004) identifies the combinatorial complexity as being related to the number of components in a system or the number of combinations of system components that are possible.

On the other hand, dynamic complexity is not obviously dependent on system's size, but a consequence of components interaction over time (Sterman, 2000). This can happen in all systems, both small and large, and when they are highly interconnected the dynamic complexity will exhibit.

It can be illustrated by a simple example, concerning a gas fuel supply chain consisting of a retailer, wholesaler and refinery. The retailer orders gas tanks from the wholesaler, who in turn orders gas from the producing refinery. Since there is a delay between placing an order and receiving the tanks, a small perturbation in the number of gas tanks sold by the retailer can cause large shifts in the quantity of tanks stored and produced by the wholesaler and refinery respectively. It is obvious that such a system is subject to dynamic complexity.

Senge (1990) emphasizes three effects of dynamic complexity:

- An action may have different effects in the short and long run.
- An action may have very different consequences in one part of the system to another.
- An action may lead to non-obvious consequences.

These effects, usually involved by the feedback connections within a system, make it very difficult to estimate its response when actions are taken or changes are made. The interconnections in operations systems are often not unidirectional, and so loop structures and feedback are quite common. In particular, physical items and information often flow in opposite directions. In a supply chain, for instance, physical items often move towards the customers while information about orders for more stock moves towards the producers. In some cases the loop structures are very complex, involving many system components, as shown by Robinson (2004).

He mentions also that many operational systems are interconnected and subject to both variability and complexity (combinatorial and dynamic). Because it is difficult to estimate the systems outputs when they are subject to any one of variability, interconnectedness and complexity, it is rather impossible to predict the output when systems are potentially subject to all three. Simulation models, however, are able explicitly to represent the variability, interconnectedness and complexity of a system. As a result, by simulation it is possible to predict system behavioral performances, to compare alternative system designs and to determine the influence of alternative operating strategies on outputs characteristics.

1.3.2. Modelling and simulation – some advantages

Robinson (2004) reveals that simulation is not the only method of studying operational systems. Rather than develop and use a model for simulation purposes, experiments could be carried out in the real system. There are some obvious, and less obvious, reasons why simulation is preferable to such direct experimentation.

- *Cost*. Real experiments are estimated to be costly, as they interfere with the normal operating regime of the studied system. New parts to be added, other materials, energy, skilled operating personnel, all of these have a particular cost, usually not negligible at all. Moreover, if the experiments cause the system's performance to worsen, this brings not only additional costs but also the customer dissatisfaction. With a simulation, however, experimentation implies only the cost of time it takes to adapt the model, as it does not interrupt the real system at all.
- *Time*. Experimenting with a real-world system is time consuming, as improving its performances is not an easy task (at least when the approach uses a *try-then-evaluate* algorithm). On the other hand, it is expected that a simulation can run faster than real time. Consequently, a re-evaluation of new system's performance can take minutes, maybe hours, but not days. Such a fast experiment enables many ideas to be explored in a short time horizon. This also

has the advantage of covering a very long time frame (i.e. months or even years of system operation) if required.

- *Control of the experimental conditions*. When experimenting with the real system it is indeed difficult to fully control the conditions under which the experiments are performed (otherwise direct comparisons cannot be made). For instance, supposing new emergency procedures have to be tested, a large-scale system crash in a power plant could not be experimented at all. But with a simulation the experimental conditions can be repeated many times, allowing the so-called *pattern-based experimentation*.
- *The real system does not exist*. This is the major difficulty with real world experimentation, when the studied system may not yet exist. In this extreme case direct experimentation is impossible, the only alternative being to develop a model.

Obviously, model-based simulations are not the only methods to be used for understanding and improving the real world systems. Other approaches range from simple paper calculations, through spreadsheet models, to more complex mathematical programming and heuristic methods (e.g. linear programming, dynamic programming, genetic algorithms – Robinson, 2004). But there are two reasons why simulation would be used in preference to these alternatives, shortly mentioned here.

- *Modelling variability*. Unlike simulation, the methods mentioned above are not even able to deal with any kind of variability or, if they do, their complexity becomes prohibitive. In this context, simulation is often the only way for pertinently estimating the studied system's behavior when it subject to significant variability.
- *Restrictive assumptions*. While other modelling approaches require certain assumptions (in order to meet their basic principles),

model-based simulation is much less restrictive. Of course, the desires to simplify models as well as data amount shortage need that some appropriate assumptions are normally made, but this does not restrict at all the simulation's principles.

Of course, there are occasions when another studying approach is appropriate and simulation is not required. More, Pidd (1998) reveals that, being a time consuming method, sometimes simulation should be used as a means of last resort, rather than the preferred option.

1.3.3. The disadvantages of simulation

There are a number of problems with using simulation and these must not be ignored when deciding whether or not it is appropriate. The most important disadvantageous characteristics (below presented) were also emphasized by Robinson (2004).

- *Expensive*. Simulation software is usually not cheap, as well as the cost of model development, implementation and use.
- *Time consuming*. Because simulation is a time consuming technique, the benefits are not immediate, while the time spent during experiments increases the total cost of the project.
- *Data hungry*. Most simulation models become useful only when a significant amount of data is available. Even more, supposing the required data set is collected, supplementary procedures may be involved in order to put it in a form suitable for the simulation.
- *Requires expertise*. Model-based simulation involves human skills in, among other things, conceptual modelling, validation and statistics, as well as working abilities when dealing with people, which are not always readily available.
- *Overconfidence*. Computer simulations results have not to be considered right "by default", only because they were produced by a good algorithm running on powerful computing platforms.

Significant effort has to be paid when interpreting the results from a simulation, as much consideration must be given to the validity of the underlying model as well as to the assumptions and simplifications that have been made.

1.4. When to Simulate

It is impossible to give a full list of applications for which simulation might be used. It is, however, interesting to give some indication of the range of systems that can be modelled. Banks et al. (1996) suggest the following items on their list, mentioning also that other applications may be added:

- Physical and chemical systems in industry;
- Business process (re)engineering/management;
- Infrastructure computer-based systems;
- Manufacturing systems (especially flexible production lines);
- Transportation systems;
- Public systems: natural resources, health care, military;
- Domestic industries (constructions, food processing).

1.5. Chapter's Conclusions

This first chapter deals with the nature of simulation, as it is involved in this course. Some specific definitions of simulation for modelling operations systems are provided, trying to cover the many meanings associated with term "simulation". By presenting the main advantages and disadvantages of model-based simulation approaches, some reasons for using simulation are also discussed. Finally, a short list of common simulation applications is presented, in order to have a complete overview on its implications and importance in real life.

2. Software Tools for Simulation

2.1. An Overview

The simulation software evolution closely follows the history of electronic computing. The first rudimentary computer simulations were performed in early years of informatics era, around 1950 (although it is to mention that, at the end of World War Two, in 1940s, a sort of military computers served for ballistic simulations).

In the next decade, programming languages such as FORTRAN greatly were involved in simulation projects. In 1960s the first specialist simulation languages such as GPSS (Schriber, 1974) and SIMULA (Dahl and Nygaard, 1966) were announced. Since these early simulations were based on computer source-code only, they appear like a black box into which data were input and results were output following a simulation run (Robinson, 2004).

As 1970s are well known for the microprocessors and microcomputers spreading, the potential of visual and interactive simulations (VIS) started to be revealed. The first language for VIS, SEE-WHY, was announced in 1979 (Fiddy et al., 1981). Consequently, in 1980s and 1990s (the PC's years) and in the last decade, a wide range of simulation languages and simulators became available (Law and Kelton, 2000). Improvements in functionality and graphical interface facilities, the compatibility with other software packages, online use (across the Internet) are only a few characteristics of the modern simulation environments.

On the other hand, this very generous range of software offers leads to another problem: how to select the appropriate tool for model development, when a simulation-based project has to be started. This chapter attempts to find appropriate answers for the next questions:

- What types of software can be used for model development?
- In this context, what specific packages are available?
- What selection criteria are appropriate?

2.2. Visual Interactive Simulation

At present, simulation models could be described as being visual interactive simulations (Robinson, 2004), meaning the model provides a visual display while simulation runs. The displayed graphical information can range from a simple schematic (as shown in figure 2.1) to a very complex animation, for instance.

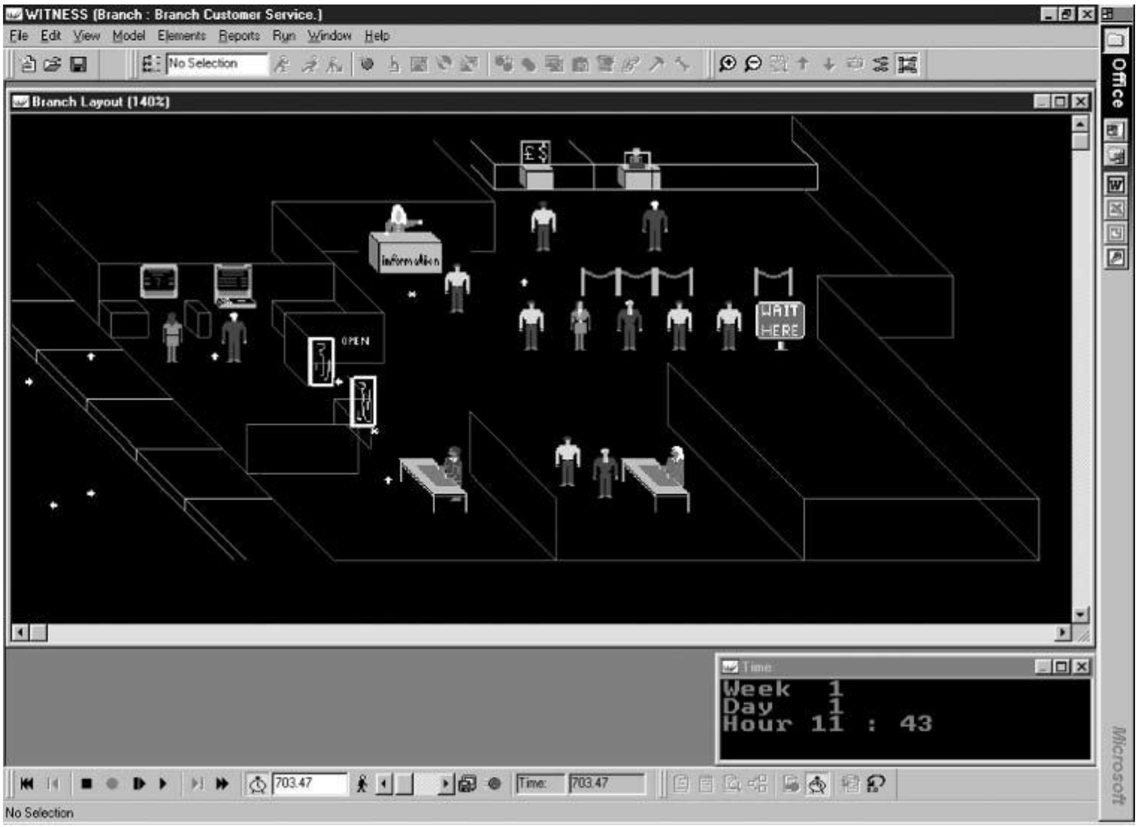


Figure 2.1. A Typical VIS Working Session.

Another VIS characteristic is the ability of interacting with the running model. The simulation can be stopped at any point in order to obtain additional information about the status of the model and the performance of the system being modelled. Even more, in some implementations the model can be altered before running it further, although the paradigm of *model invariance* is also well-respected in the scientific community. The simulation can also stop at a point when it requires dialogue with the user. Taking into account these considerations, Robinson summarizes the main benefits of VIS as follows:

- *Greater understanding of the model*. The visual display allows the model user to track all events occurring in the simulation, which may be similar with the corresponding real system.
- *Easier model verification and validation*. The modelling errors can be identified by the non-usual events arising during simulation and, with the help of non-simulation experts (but having good knowledge about the modelled system), they can be corrected.
- *Enables interactive experimentation*. Providing a faster response than real time, simulations allow new ideas testing as they are issued. This way, the understanding of the model and the operational system is seriously improved.
- *Improved understanding of the results*. By VIS, it becomes easy to identify logical connections between results and specific events observed during simulation. Re-running the model also makes possible to understand why specific results have been obtained.
- *Improved dissemination of the model*. By VIS techniques, non-simulation experts are able to understand the model, enabling a wider group to access a simulation-based project.
- *Provides the potential for group problem solving*. Validation and experimentation can be carried out in a group setting with input

from a range of interested parties. This can facilitate greater creativity and consensus in problem solving (Robinson, 2001).

2.3. Simulation Software

At present, there are three options for developing computer modelling and simulation software tools: spreadsheets, general programming languages and dedicated environments.

2.3.1. Spreadsheets

Spreadsheet packages (i.e. Excel) have some basic capabilities for modelling and simulation. For instance, it is relatively straightforward to develop a simple mathematical model describing the input-output dependency for a simple system (in stationary regime). But, beyond this rudimentary level, it is necessary to use some programming capabilities within the spreadsheet, like macros or Visual Basic code when using Excel. As this context requires adequate programming skills, it may be useless and time consuming to develop a spreadsheet-based simulation tool instead of natively using the programming languages, as shown in the next paragraph.

2.3.2. General programming languages

A great flexibility can be reached when simulation models and environments are developed by using general purpose programming languages such as Visual Basic, C++, Java and so on. But, on the other hand, all simulation capabilities have to be built from scratch, revealing a big drawback of this approach (as it proves to be time consuming). Fortunately, modern languages use the object oriented programming technique which allows the code encapsulation and re-use; this manner of work can reduce the developing effort when standard "objects" (simple models, numerical methods, graphic routines) are collected in portable libraries.

2.3.3. Dedicated software environments

These specialist simulation software packages are widely available today. Law and Kelton (2000) identify two types of dedicated simulation environments:

- *General purpose simulation packages*, serving a wide range of applications (MATLAB, for instance). They are generously covering almost everything which may be simulated, but require high skills when implementing a particular application.
- *Application orientated simulation packages*, dealing with specific cases (i.e. chemical processing – PRO/II, analog/digital electronics – Pspice, ORCAD). These focused packages are easier to use, but they serve a narrower range of application.

Almost all of these software packages, but especially the dedicated environments, could be described as visual interactive modelling systems (VIMS) (Pidd 1998). They enable a visual and interactive manner of building a simulation application. VIMS software provides a predefined set of simulation objects which can be used to define the model logic, the simulation engine and the user interface through a set of menus. As result, much more reduced programming skills are usually required. A different situation appears when VIMS use an embedded programming language, in order to deal with very complex modelling applications. Obviously, adequate programming skills are required in this case.

As remark, Robinson (2004) reveals that VIS and VIMS terms should not be confused. VIS deals with the intrinsic model nature while VIMS refers to how it is built. Indeed, a VIS can be developed without using a VIMS (a general-purpose programming language can be used instead, for example). On the other hand, a simulation model built using a VIMS may not have a visual display, so it is not a VIS.

2.3.4. Comparing spreadsheets, programming languages and specialist simulation software

Table 2.1, adapted from Robinson (2004) presents a schematic comparison between the simulation modelling approaches described above, giving just an idea on their advantages and disadvantages.

Table 2.1. A comparison between spreadsheets, general programming languages and dedicated simulation environments

Feature	Spreadsheets	Programming languages	Dedicated software
Range of application	Low	High	Low/medium
Modelling flexibility	Low	High	Medium
Implementation time	Medium	Long	Short
Ease of use	Medium	Low	High
Ease of model validation	Medium	Low	High
Performance (speed)	Low	High	Medium
Time to obtain software skills	Short/medium	Long	Medium
Price	Relatively low	Low	High

It shows that general programming languages serve the widest range of applications, also having the most flexible behavior when implementing the model. Meanwhile, developing in programming languages leads to a shorter execution time than equivalent implementations (by spreadsheets or dedicated environments). On the other hand, the model build within specialist simulation software is relatively easy and takes a shorter time. Spreadsheets are sometimes better than programming languages in respect of model build speed and ease of use (at least for smaller applications), but they are not as quick or straightforward to use as the specialist software (Robinson, 2004).

However, the time required for getting appropriate skills increases if the macro language is used for model development. Regarding the costs, spreadsheets and programming languages are similarly priced (at low or moderate values), while dedicated simulation software tends to have the biggest cost – which may be correct as time as it is very productive as developing tool.

The software selection critically depends upon the simulation study characteristics (especially its complexity). While for very simple applications a spreadsheet may be the best option, usual applications, with pertinent complexity, ask for more powerful software. Dedicated (general purpose) simulation packages are able to model a wide range of applications and can be used as time as the model is not highly complex. In this last case it is expected that only a programming language is appropriate.

As remark, this course assumes that a specialist simulation package is used by the developing team, since there are big chances it is suitable for modelling most operational systems.

2.4. Selecting the Simulation Software

Once the nature of simulation software was briefly presented, this section deals with the problems which may be taken into account when selecting an appropriate solution for the (dedicated) modeling and simulation environment.

The importance of software package selection for a successful simulation project implementation has been emphasized by some authors (Law and McComas, 1989, for instance), while others minimize this dependency (Robinson and Pidd, 1998). The truth is that many researchers repeatedly use the same simulation environment on different problems, always trying to adapt the software to meet the project requirements. For them, the advantage of using an already available and familiar tool seems to be above all selection criteria. But how can be explained this apparent difference in view? A possible answer was synthesized by Robinson in 2004 as follows. *Within a certain domain of application, most of the "more powerful" simulation packages are quite capable of modelling what is required. Indeed, with their growing capabilities over the past years, this domain of application has steadily increased. There are always, of course, applications that go beyond the capabilities of a software package. It is when we are dealing with these applications that careful software selection is needed. Because much of the available software has been designed specifically for modelling operational systems, there are few occasions on which the software simply cannot model these systems. As a result, software selection becomes more a matter of the convenience with which the system can be modelled than the capability of modelling the system. It has to be said that as long as the software suffices, the expertise of the modeller (e.g. in problem solving, statistics, project management, people management and communication) is probably of far greater importance to successful simulation modelling*.

2.4.1. The process of software selection

Some authors describe, in the open literature, a series of steps for selecting simulation software – Holder (1990), Hlupic and Paul (1996), Nikoukaran and Paul (1999), Bard (1997). Although there are some differences between them, the selection process could be summarized as follows (Robinson, 2004):

- **Step 1**: establish the modelling requirements
- **Step 2**: survey and shortlist the software
- **Step 3**: establish evaluation criteria
- **Step 4**: evaluate the software in relation to the criteria

- **Step 5**: software selection

Usually this seems to be a linear process (from step 1 through to 5), with possible iterations between steps 3 to 5, as it will be shown below.

2.4.2. Step 1: Establish the modelling requirements

When establishing these requirements, some aspects have to be also taken into account. First, the nature of the systems to be modelled should be identified. Then, the software future utility (singular or general use) and application (focused on a narrow or wide domain of application) have to be decided. The modelling type is also important, as time as a simple approach requires ease-of-use, while complex/detailed modelling needs a high level of functionality (Hlupic and Paul, 1996). Of course, any other constraints have to be taken into account (i.e. finance availability, existing personnel skills, and hardware/software policy of the client organization).

2.4.3. Step 2: Survey and shortlist the software

After finishing step 1, the next task is to create a short list of appropriate software. Starting from a "complete" list, the short one can be written by obtaining outline information on the software to determine whether they meet the modelling requirements (usually from vendor web sites). Further to this, the critical surveys as well as experts advices carried out in the open literature can provide some useful information (Robinson, 2004). All these can quickly eliminate the packages not following the established requirements.

2.4.4. Step 3: Establish evaluation criteria

Criteria for comparing the chosen simulation packages need also to be established. Table 2.2, adapted from Robinson (2004) provides a list of criteria, grouped under a series of main objectives. Obviously, not every criterion from these lists should be included in the evaluation, but only the selected ones, which fit the modelling requirements from step 1.

Table 2.2. Some Criteria for Simulation Software Selection

Main objectives	Criteria to be taken into account
Hardware/software requirements	Hardware platform required Operating system required Software protection (hardware keys) Availability of network licenses Features for use on the World Wide Web
Model coding and testing	Ease of model development Possibility to build and run models in small steps Availability of debugging tools Maximum model size Maximum dimensions of objects (e.g. arrays) Features for documenting a model Availability of help facility Availability of software wizard
Visual features	Online/offline results analysis on the display Speed with which display can be developed Customizable user icons Availability of icon libraries Ability to pan and zoom Ability to locate objects on the display Smoothness of animation Availability of 3D animation
Input data and analysis features	Distribution fitting Ability to sample from empirical distributions Availability of statistical distributions Ability to import data from other software
Reporting and output analysis features	Availability of standard reports for model objects Availability of graphical reporting Ability to develop customized reports Ability to export results to other software Statistical analysis of results
Experimentation	Probable run-speed Run control (step, animated, batch) Interactive capability Number of random number streams available Control of random number streams Ability to perform multiple replications Facilities for organizing batches of runs Provision of advice on warm-up, run-length and multiple replications Availability of an optimizer Ability to distribute runs across networked computers

Main objectives	Criteria to be taken into account
Support	Availability of a help desk Availability of consultancy support Type of training given Frequency of software upgrades Foreign language versions and support Quality of documentation
Pedigree	Size of vendor's organization The package age/maturity References on similar applications using the package Number of users (in industry sector) Geographic usage of the package Availability of literature on the package and its use
Cost	Purchase price Maintenance fee Cost of support Cost of training Time to learn the software Availability of lower cost run-only license

2.4.5. Step 4: Evaluate the software in relation to the criteria

Each of the chosen packages needs to be evaluated through the selected criteria, by employing means like (Robinson, 2004):

- Discussion with the software vendor and other users of the software
- Software and model demonstrations
- Obtaining a free evaluation copy of the software
- Studying the software documentation and other literature
- Asking for expert opinion

Of course, one should take into account the time available for the evaluation, as time as – for instance – any approach that requires the development of models is going to require significantly more time.

The evaluation should lead to an assessment of the extent to which each package meets the criteria previously set out. Sometimes, a simple "yes" or "no" by each criterion to indicate whether or not a package has that capability may be enough. However, when some degrees of capability are involved, it is better to use a scoring scale (for 1 to 10, for instance) indicating the compliance level. Some criteria can be assessed objectively (e.g. purchase price), while for others subjective judgments must be made (e.g. quality of documentation). As far as possible, it is best to identify significant measures to evaluate the criteria (Robinson, 2004).

2.4.6. Step 5: Software selection

A specific package can be selected based upon the extent to which it meets the chosen criteria. This may be based on a subjective judgment (a simple comparison of the package evaluations – step 4) or can be more objective when using an overall score. Because each criterion does not have the same level of importance, Robinson (2004) suggests it is useful to weight the criteria according to their objective/subjective importance (in fact, the weighting factors need to be obtained from key members of the client organization).

Then, an overall score could then be calculated for each package as follows:

$$S_i = \sum_j W_i E_{ji}$$

where *S_i* is the overall score for software package *i*, *W_j* – importance weight for criterion *j*, *E_{ji}* – evaluated score for criterion *j* for package *i*.

A special case is when, having a large number of criteria, it is impossible to assign the *W_i* weights in a consistent way. This issue may be addressed through the Analytic Hierarchy Process (AHP) technique, described by Saaty (1980). Although its scope exceeds the purpose of this course, the interested reader may follow a typical example of using AHP for selecting simulation software, presented by Davis and Williams (1994).

2.5. Chapter’s Conclusions

This second chapter focuses on simulation software characteristics, giving also some basic indications on how to select such a package. All following the same model building/running principle (VIS – Visual Interactive Simulations), three types of software are available for developing simulations: spreadsheets, general programming languages and dedicated simulation software. The software choice depends upon the complexity of the simulation being performed.

When using specialist simulation software, the user can benefit from a visual interactive modelling system (VIMS) that integrates predefined sets of objects and programming interfaces oriented on model building and running. The process of selecting such a package involves the establishment of modelling requirements, packages primary selection and the final decision (taken by evaluating the criteria reflecting the needs of the client organization).

3. The Conceptual Modelling Principles

3.1. Introduction

A simulation of a power plant could take many forms. At the simplest level the model might include only the generators (seen as "black boxes") and their supervisory system. The model, however, could be expanded to include the fuel supply system, operating personnel behavior and site management structure. There is also a need to consider the level of detail at which each sub-system has to be modelled (the generators, for instance, could be assumed as having fixed characteristics or following a statistical distribution). At a greater level of detail, a single generator could be modelled through an input-state-output approach (with equations describing its inner phenomena). Also, the process failures and interruptions could be modelled. The modeller, along with its clients, must determine the appropriate scope and level of detail to model, a process known as *conceptual modelling* or *designing the model*. In this context, this and the next chapter describe the requirements for conceptual modelling, also presenting how a simulation specialist might go about designing the conceptual model. In this chapter the importance of conceptual modelling is emphasized before defining the term *conceptual model* more precisely. The requirements of a conceptual model are then described. Finally, the practical issue of how to communicate the conceptual model to all members of the simulation project team is discussed. In the next chapter, the question of how to design the conceptual model is covered.

3.2. The Importance of Conceptual Modelling

Conceptual modelling is almost certainly the most important aspect of the simulation modelling process (Law, 1991). The model design impacts all aspects of the study, in particular the data requirements, the speed with which the model can be developed, the validity of the model, the speed of experimentation and the confidence that is placed in the model results. A well designed model significantly enhances the possibility that a simulation study will meet its objectives within the required time-scale. What sets truly successful modellers apart is their effectiveness in conceptual modelling (Ward, 1989).

Robinson (2004) has its own point of view: *It is often said of simulation studies that 50% of the benefit is obtained just from the development of the conceptual model. The modeller needs to develop a thorough understanding of the operations system in order to design an appropriate model. In doing so, he/she asks questions and seeks for information that often has not previously been considered. In this case, the requirement to design a simulation model becomes a framework for system investigation that is extremely useful in its own right. Indeed, Shannon (1975) goes so far as to say that effective conceptual modelling may lead to the identification of a suitable solution without the need for any further simulation work.*

But some might argue that the emergence of modern simulation software has reduced, or even removed, the need for conceptual modelling. After all, the specialist can now move straight from developing an understanding of the real world problem to creating a computer model. What this point of view ignores is that the modeller still has to make decisions about the content and assumptions of the model. Of course, modern simulation software provides an environment for more rapid model development, making prototyping more feasible and enabling a greater level of iteration between conceptual modelling and computer modelling. But the software does not, however, reduce the level of decision-making about the model design.

On the contrary, it could be said that the power and memory of modern hardware and the potential for distributed software have increased the need for conceptual modelling. Salt (1993) and Chwif et al. (2000) cite in this context the "possibility" factor: *People build more complex models because the hardware and software enables them to. Although this extends the utility of simulation to problems that previously could not have been tackled, it is also likely that models are being developed that are far more complex than they need be. In this sense there are certain advantages in having only limited computing capacity; it forces the modeller to design the model carefully! As a result of the extended possibilities, careful model design is probably increasing in importance.*

Although conceptual modelling is of utmost importance, it must also be recognized that it is probably the least understood aspect of simulation modelling (Robinson, 2004). There is surprisingly little written on the subject in the open literature. The main reason for this lack of attention is no doubt that conceptual modelling is more of an "art" than a "science" and therefore it is difficult to define methods and procedures.

So, the "art of conceptual modelling" is largely learnt by experience. This is not a satisfactory situation for such an important aspect of the simulation modelling process. In order to address this issue, this and the next chapter attempt to provide specific advice on how to develop a conceptual model. This is done by looking at the subject from various angles. This chapter introduces the basic concepts of conceptual modelling. First, the meaning of conceptual modelling is more precisely defined. Then the requirements of a conceptual model are discussed. The chapter concludes by discussing the reporting and communication of the conceptual model. Chapter 4 goes on to discuss the actual process of conceptual modelling and how the conceptual model is designed.

3.3. The Definition of a Conceptual Model

Zeigler (1976) tries to clarify the definition of a conceptual model by distinguishing between four terms: *the real system* is that which the simulation model is to represent. *The experimental frame* is the limited set of circumstances under which the real system has been observed, in other words, there is not a complete understanding of the real system. The full-scale model is capable of accounting for the complete behavior of the real system, but, since this model is very complex, it cannot be completely known. Meanwhile, in the reduced-scale model the system components and their interconnections are simplified. The structure of this model is fully known to the specialist. In our terms, the reduced-scale model (known as *lumped model* in the literature) and conceptual model may be considered equivalent (Robinson, 2004).

This definition, however, provides little more than a sense that the conceptual model is a simplified representation of the real system. A more descriptive definition of a conceptual model is as follows:

The conceptual model is a non-software specific description of the simulation model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplifications of the model (Robinson, 2004).

There are two key features of this definition. First, it specifically identifies the independence of the conceptual model from the software in which the simulation is to be developed. Indeed, in an ideal world the software should be selected on the basis of the conceptual model understanding.

Since the world is less than ideal, it is often the case that the conceptual model is designed around the software that is available to the modeller.

The second feature is that the definition outlines the key components of the conceptual model, which are as follows:

- *Objectives*: the purpose of the model and modelling project.
- *Inputs (experimental factors)*: those elements of the model that can be altered to effect an improvement in (or better understanding of) the real world.
- *Outputs*: report the results from simulation.
- *Content*: the components that are represented in the model and their interconnections.
- *Assumptions*: statements made when there are uncertainties about the real world being modelled.
- *Simplifications*: statements on which the model is based, in order to enable more rapid model development and use.

Assumptions and simplifications are different. Assumptions are ways of incorporating uncertainties and beliefs about the real world into the model. Simplifications are ways of reducing the complexity of the model. As such, assumptions come from limited knowledge or presumptions, while simplifications focus on the desire to create simple models (Robinson, 2004).

The content of the model should be described in terms of two dimensions (Robinson, 1994):

- *The scope of the model*: the model boundary or the real system dimensionality that is to be included in the model.
- *The level of detail*: the detail to be included for each component in the model's scope.

The purpose of the conceptual model is to set out the basis on which the computer based simulation (computer model) is to be developed. It is in effect a functional specification of the computer software. For many modelling specialists there is a temptation to start coding the computer model as soon as possible. Without due attention to the development of the conceptual model, however, this can lead to a model that does not achieve what is required and, at the extreme, the model may have to be completely rewritten, wasting significant amounts of time (Robinson, 2004).

3.4. Requirements of the Conceptual Model

In designing a conceptual model it is useful to establish a set of requirements. In this way the model can be designed so as to meet these requirements.

3.4.1. The four requirements

Willemain (1994) emphasizes five qualities of an effective model: *validity*, *usability*, *value to client*, *feasibility* and *aptness for clients' problem*. Meanwhile, Brooks and Tobias (1996) identify 11 performance criteria for a good model. Based on these lists, here it is proposed that there are four main requirements of a conceptual model:

- Validity
- Credibility
- Utility
- Feasibility.

A valid model is one that is sufficiently accurate for the purpose at hand. However, since the notion of *accuracy* is of little meaning for a model that has no numeric output, conceptual model validity might be defined more precisely as *a perception, on behalf of the modeller, that the conceptual model will lead to a computer model that is sufficiently accurate for the purpose at hand* (Robinson, 2004).

Underlying this notion is the question of whether the model is right. This definition places conceptual model validity as a perception of the modelling specialist. It also maintains the notion that a model is built for a specific purpose, which is common to most definitions of validity.

Credibility is similar to validity, but is taken from the perspective of the clients rather than the modelling specialist. The credibility of the conceptual model is therefore defined as *a perception, on behalf of the clients, that the conceptual model will lead to a computer model that is sufficiently accurate for the purpose at hand* (Robinson, 2004).

The third concept, *utility*, is defined as *a perception, on behalf of the modeller and the clients, that the conceptual model will lead to a computer model that is useful as an aid to decision-making within the specified context* (Robinson, 2004).

Whereas the definitions of validity and credibility are specific to the modelling specialist and the clients respectively, utility is seen as a joint agreement about the usefulness of the model. The concept of utility, as defined here, moves away from simply asking if the model is sufficiently accurate, to whether it is useful. Within any context a range of models could be designed, all of which might be sufficiently accurate for the purpose at hand. As such, all these models would be valid and credible. However, if a proposed model is sufficiently accurate, but too large and stiff, it may have limited utility. Indeed, a less (but still sufficiently) accurate and more flexible model that runs faster may have greater utility by enabling a wider range of experimentation within a given time-frame. The final requirement, *feasibility*, is defined as *a perception, on behalf of the modelling specialist and the clients, that the conceptual model can be developed into a computer model* (Robinson, 2004).

Various factors may make a model infeasible. For instance, it might not be possible to build the proposed model within the required time-scale, the data requirements of the model may be too high, or there is insufficient knowledge of the real system to develop the proposed model.

A final point to note is that these four concepts are not mutually exclusive. A specialist's perception of a model's accuracy is likely to be highly correlated with the clients' perceptions of the same. On the other hand, an infeasible model is not a useful model. It is good idea, however, to separate these concepts, so a specialist can independently use them when designing the conceptual model (Robinson, 2004).

3.4.2. Simplicity – a key feature

In the context of avoiding an over-complex model development, the aim should be to keep the model as simple as possible to meet the objectives of the simulation study (Robinson, 2004). Simple models have a number of advantages. They can be developed faster, are more flexible, require less data, run faster, and it is easier to interpret the results since the structure of the model is better understood (Innis and Rexstad, 1983; Ward, 1989; Salt, 1993; Chwif et al., 2000). As the complexity increases these advantages are lost.

Keeping models simple is a basic principle of good modelling practice. This does not mean that complex models should never be developed, because they are sometimes necessary to achieve the objectives of the study. There is, however, a tendency to try and model every aspect of a system when a simpler, more focused model would achieve the modelling objectives with far less effort.

On the other hand, it is impossible to create a model that is 100% accurate, since it is not possible to capture every aspect of the real world in a model. Indeed, increasing the complexity too far may lead to a less accurate model, since the data and information are not available to support the details being modelled.

Ward (1989) makes a useful distinction between *constructive simplicity* and *transparency*. Transparency is an attribute of the client (how well he/she understands the model) while constructive simplicity is an attribute of the model itself (the simplicity of the model). The modelling specialist must not only consider simplicity, but also transparency in designing a model. Since transparency is an attribute of the client, it is dependent on the client's knowledge and skill. In other words, a model that is transparent to one client may not be to another. The specialist must, therefore, design the model with the needs of the particular client in mind. This is necessary to develop the credibility of the model as previously discussed, since a model that is not transparent to the client is unlikely to have credibility.

3.5. The Conceptual Model Dissemination

In order to determine whether the conceptual model meets the four requirements already set out, it is important that there is a shared understanding of the modelling context (real world) and model design between the modelling specialist and clients (as well as the other roles in the simulation study). In this context, a mechanism for communicating the conceptual model is strongly required, as part of the project specification.

3.5.1. Project specification

The output from conceptual modelling should be described in a project specification along with details of how the simulation study is to be managed. In this way a shared understanding of the conceptual model and the simulation study can be developed between all project team members. Indeed, the project specification acts as the primary means for validating the conceptual model. It also provides a reference point for developing and verifying the computer model, performing appropriate experiments and reviewing the success of the simulation study.

Depending on the nature of the project and the relationship between the clients and modelling specialist, the specification should describe the majority, if not all, of the following items (Robinson, 2004):

- Background to the problem situation
- Objectives of the simulation study
- Expected benefits
- The conceptual model: inputs, outputs, content (scope and level of detail), assumptions and simplifications
- Experimentation: scenarios to be considered
- Data requirements: data required, when required and responsibility for collection
- Project time-scale
- Estimated cost.

In general the specification takes the form of a written document that can be circulated to all involved in the simulation study. If possible, it is best to keep the document fairly short, to ensure that it is read and valuable feedback is obtained. But, in fact, all depends on the size and complexity of the model and the formality of the process required.

It is vital that the specialist obtains feedback, so being able to judge the validity, credibility, utility and feasibility of the proposed model. There should also be some discussion about the management of the project, for instance, the collection of data, time-scales and costs. To aid this process, it may be useful formally to present the project specification to the simulation project team and to obtain immediate feedback (Robinson, 2004). This is particularly necessary when assumptions and simplifications are questioned. The modelling specialist must decide to change the model or justify the assumption or simplification. The judgment as to which depends on the extent to which a change versus a justification affects the validity, credibility, utility and feasibility of the model.

Because any simulation study is an iterative process, it should not be expected that once model coding is started the specification remains unchanged. There are four main reasons why it should be expected that the specification will change during a simulation study (Robinson, 2004):

- Omissions in the original specification
- Changes in the real world
- An increased understanding of simulation on behalf of the clients
- The identification of new problems through the development and use of the simulation model.

Effective conceptual modelling, communication and feedback should limit the first cause of change. Changes to the real world inevitably happen, for instance, a change to the design of a power plant that may be on a small scale (e.g. an additional generator) or on a larger scale (e.g. a complete redesign). The last two reasons for change are both positive aspects of simulation modelling and should be encouraged.

Because things change, it is important that a mechanism is put in place for handling these changes. If the model is simply updated without any proper reporting, then the specification soon becomes outdated and there is no way to trace the model alterations. To maintain a record of changes, it is useful to have a "specification change form" that is used every time an alteration is made (Robinson, 2004). This can be circulated to ensure all are informed and agree to the change.

Of course, if the conceptual model is continuously changing, it may become impossible to complete the model development and experimentation. It is useful, therefore, to reach a point where it is agreed that the specification is fixed. From this point on, all change issues are logged, but unless the change is particularly significant, the conceptual model is not altered. Once the simulation is complete and the results have been reported, a further run of the simulation process may be carried out with the logged changes included in the model. The need for this depends on whether the changes are judged to be of sufficient significance to warrant further modelling.

3.5.2. Methods of representation

As part of the project specification it is important to have a means for representing the content of the conceptual model. There are four main methods of representation in common use (Robinson, 2004):

- Component lists
- Process flow diagrams
- Logic flow diagrams
- Activity cycle diagrams.

It is not the intention of this course to provide detailed descriptions of these model representation methods, but the interested reader may consult the related literature for additional information.

Of course, more than one of these methods may be used to give a different view of the same conceptual model. There are also some other methods of conceptual model representation, for instance, Petri nets (Torn, 1981), event graphs (Som and Sargent, 1989) and condition specification (Overstreet and Nance, 1985). UML (the Unified Modeling Language) is currently used for representing a conceptual model (Richter and Marz, 2000). Meanwhile, Pooley (1991) gives an overview of diagramming techniques that might support conceptual modelling.

3.6. Chapter's Conclusion

Conceptual modelling is almost certainly the most important aspect of a simulation study. It is vital that an appropriate model is designed in order for the rest of the simulation study to succeed. Unfortunately, conceptual modelling is also the least understood aspect of simulation modelling. This course chapter addresses the issue by providing a definition for a conceptual model and describing the requirements of a conceptual model (validity, credibility, utility and feasibility). It is important to design a model that is as simple as possible, while ensuring that it can meet the objectives of the study. The use of a project specification for communicating the conceptual model and methods of representing the model are also described. As consequence, the attention finally turns to the process of designing the conceptual model.

4. The Conceptual Model Aggregation

4.1. Introduction

The previous chapter provided the basic concepts behind conceptual modelling, in particular, the definition and requirements for a conceptual model. This chapter focuses on how to develop a conceptual model, from two perspectives. First, a framework for developing a conceptual model is described. Secondly, some methods of model simplification are discussed, in order to improve its quality, usability and significance.

4.2. Modelling in a Conceptual Framework

The process of designing a conceptual model is seen as "art", so there is very little guidance available. The most useful one may come from those who have offered a set of modelling principles (Morris, 1967; Powell, 1995; Pidd, 1999). These range from the socio-political, such as regular contact with subject matter experts, to the more technical, such as developing prototype models along the way. Although these principles are useful for giving some guide to conceptual model design, they do not answer the question of *how to develop the conceptual model*.

Figure 4.1 (Robinson, 2004) provides a general framework for conceptual modelling. The purpose of this framework is to provide a modelling specialist with an understanding of how to develop a conceptual model. The framework consists of four key elements:

- Develop an understanding of the problem situation
- Determine the modelling objectives

- Design the conceptual model: inputs and outputs
- Design the conceptual model: the model content.

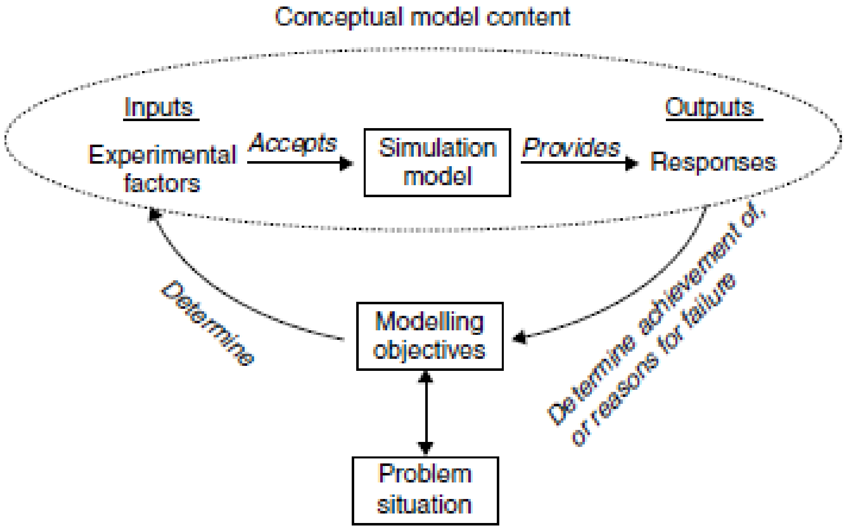


Figure 4.1 A Framework for Conceptual Modelling.

Starting with an understanding of the problem situation, a set of modelling objectives are determined. These objectives then drive the derivation of the conceptual model, first by defining the inputs and outputs, and then by defining the content of the model itself. These elements are described in detail below.

Before going on to detailed descriptions, it is worth remembering that in the same way that the process of performing a simulation study is iterative, so too is conceptual modelling. There is likely to be a great deal of iteration between the elements in the conceptual modelling framework, as well as with the other processes in a simulation study (Robinson, 2004). Some of the reasons for this iteration are discussed in the description that follows.

In order to illustrate the framework utility, an example of modelling the same power plant is used. This context has been chosen since it should be already familiar to the reader. More, this case study is referred to throughout the rest of the course and it is suggested that the reader follow these as a means of seeing how the modelling principles are applied.

4.2.1. Understanding the problem

It is obviously necessary for the modelling specialist to develop a good understanding of the problem situation when developing a model that adequately describes the real world. The approach to this process depends in large measure on the extent to which the clients understand and are able to explain the problem situation.

In many circumstances the clients will be able to provide such an explanation, for instance, by describing the operation of the (proposed) real world system that is at the heart of the problem situation. The accuracy of the description, however, may be dubious (Robinson, 2004).

One issue is that the clients may not have a good understanding of the *cause-effect* relationships within the problem situation. For instance, in a modelling study of a power plant site, the belief may be that technical assistance department is understaffed (cause) which resulted in poor plant service (effect). Although the effect was correctly identified (and was in fact the reason why the study is performed), it has been observed that increasing staff resources provides almost no benefit in terms of improved technical assistance. What is required is a change to the human resources supervising process.

Another problem for the modelling specialist is that the clients almost certainly have different world views (Checkland, 1981). In the above study, it may seem there are as many different descriptions of how the maintenance technicians go about their tasks as people who are interviewed. This should be no surprise, especially when dealing with human activity systems in which the human behavior and decision-making process impact on the performance of the system.

What becomes apparent is that the modelling specialist has to play a much more active role. Providing the right prompts and speaking with the right people is vital to developing this understanding. The specialist should also suggest alternative versions of the events in order to facilitate new ways of perceiving the problem situation (Robinson, 2004). Such discussions might be carried out face-to-face in meetings and workshops, or remotely by telephone or email, for example.

When the clients have a reasonable image of the problem situation then discussion and careful note-taking should be enough. In addition, it is important that the modelling specialist confirms his/her understanding by providing descriptions of the problem situation for the clients. This acts as a means of validating the conceptual model as it is developed (Robinson, 2004). If the clients have a poor image of the problem situation, then more formal problem structuring methods may be useful, for instance, soft systems methodology (Checkland, 1981), cognitive mapping (Eden and Ackermann, 2001) and causal loop diagrams (Sterman, 2000). Balci and Nance (1985) describe a methodology for problem formulation in simulation modelling that includes developing an understanding of the problem situation, as well as objective setting and verification of the formulated problem.

It is during the process of understanding the problem situation that areas where there is limited knowledge of the operational system are likely to be identified and so assumptions have to be made. These should be documented and recorded too in the project specification, as previously described. In fact, these areas of limited knowledge will continue to be identified as a simulation study progresses. This means that new assumptions need to be made and then added to the project specification (Robinson, 2004).

The problem situation and the understanding of it should not be seen as static, because both will change as the simulation study progresses. A simulation model and the information required to develop it almost always act as a focus for clarifying and developing a deeper understanding of the real world system that is being modelled. This acts to increase the level of iteration between modelling processes across a simulation study, with adjustments to the conceptual model being required, even at the point when the model is being used for experimentation, as new facets of the problem situation emerge (Robinson, 2004).

As stated earlier, the conceptual modelling framework is illustrated with an example of a power plant. Table 6.1 describes the problem situation at the plant.

Table 4.1. Power Plant Illustration – The Problem Situation

The Problem
A power plant is experiencing problems with one of the branches in its energy distribution network. Customers regularly complain about the length of time they have to wait for service personnel when a power failure occurs. It is apparent that this is not the result of shortages in fuel supply (for the generators), but a shortage of technical staff.

4.2.2. Identifying the modelling objectives

The modelling objectives are central to the modelling process. They are the means by which the nature of the model is determined, the reference point for model validation, the guide for experimentation, and one of the metrics by which the success of the study is judged. Later it is shown how the objectives can be used to help design the conceptual model.

Robinson (2004) has a very interesting point of view: *A model has little intrinsic value unless it is used to aid decision-making, and so the purpose of a modelling study is not the development of the model itself. If it were, then having developed the model, the objective would have been met and the study would be complete. The logical conclusion to this process is the existence of models that have never served any useful purpose, or models that are looking for a problem to solve. There are exceptions to this rule of course. For instance, a generic model of a hospital emergency unit may be developed with a view to selling the model to numerous hospitals. On the surface, the purpose of the original modelling project is the development of a model. Underlying this, however, the model developers must have in mind some purpose for the model, for instance, to determine resource requirements. Indeed, many military models are apparently developed in this fashion. A model is developed and then an application for the model is sought. In this paradigm, the model needs to be assessed whenever a new purpose is found* (Gass 1977).

In forming the objectives, a useful question to ask is "*by the end of this study what is going to be achieved?*" Beyond this, three aspects should be considered. First, *what is it that the clients wish to achieve?* Typically this involves increasing throughput, reducing cost or improving customer service. Improving the clients' understanding of the real world system, or reducing the risk of an investment may be also considered as valid objectives.

Secondly, *what level of performance is required?* To state that the objective is to increase throughput is insufficient. *By how much should the throughput be increased?* Whenever it is possible, targets of performance for each objective should be identified. These might be expressed as straightforward targets (e.g. increase/reduce by a percentage or absolute amount) or the need to optimize (i.e. maximize or minimize) some measure. Of course, this can only be done when the objective can be quantified (Robinson, 2004).

Finally, *what constraints must the clients (or modelling specialist) work within*? Often there is a limited budget or a limited number of approaches available for achieving the objectives. For instance, the clients may only be willing to consider changes in production scheduling to gain throughput improvements, while ruling out the purchase of additional equipment.

It must be recognized that the clients may not be able to give a complete set of objectives, for the same reasons as their understanding of the problem situation may be incomplete (Robinson, 2004). Further to this, the clients may have a limited understanding of what a simulation model can do for them, particularly if they have not been involved in simulation studies previously. Therefore, it is important that the modelling specialist is able to suggest additional objectives as well as to redefine and eliminate the objectives suggested by the clients. The specialist should also educate the clients, explaining how simulation might act as an aid. One means for achieving this is to demonstrate one or more models of similar problem situations, and to provide descriptions of the modelling work that underlay them. In this way the clients will obtain a better understanding of how simulation can or cannot help (Robinson, 2004). Objective setting should involve the clients in learning about simulation and its potential, as much as the modeller in learning about the problem situation. In this way the modelling specialist is able to manage the expectations of the clients, aiming to set them at a realistic level (Robinson, 2004).

Since the problem situation and the understanding of it can change, so too can the objectives. Added to this, as the clients' understanding of the potential of simulation improves, as it inevitably does during the course of the study, their requirements and expectations will also change. Consequently the iteration between the modelling processes is further increased, with changes in objectives affecting the design of the model, the experimentation and the outcomes of the project. It is for this reason that there is a two-way arrow between the *problem situation* and the *modelling objectives* in Figure 4.1.

The modelling objective for the power plant example is given in Table 4.2.

General project objectives

In designing a simulation model the modelling objectives are not the only concern. The modeller should also be aware of some more general project objectives. Time-scale is particularly important. If there is only a limited time available for the project, then the modeller may be forced into a more conservative model design. This helps reduce model development time and quicken its run-speed, reducing the time required for experimentation.

Table 4.2. Power Plant Illustration – Modelling Objectives

Modelling Objectives
The number of service staff required during each period of the day to ensure that 95% of customers wait less than one hour for service. Due to space constraints, a maximum of ten service staff can be employed at any one time.

The modeller should also clarify the nature of the model required by the clients, specifically in terms of the visual display and the type of model use. What level of visual display is needed? Is a simple schematic sufficient, or is a 3D view required? Do the clients wish to use the model themselves? If so, what data input and results viewing facilities do they require? What level of interactive capability is necessary to enable appropriate experimentation? All these issues have an impact on the design of the simulation model.

4.2.3. The conceptual model design: inputs and outputs

The first stage of conceptual model design does not involve the details of the model itself, but the model's inputs and outputs, depicted as the experimental factors and responses in Figure 4.1. It is much easier to start by giving consideration to these, than to the content of the model (Robinson, 2004). Indeed, it should be a fairly straightforward task to move from the modelling objectives to the experimental factors. In effect, these are the means by which it is proposed that the objectives are to be achieved.

Although the clients would often have control over the experimental factors in the real world, it is sometimes useful to experiment with factors over which they have little or no control (e.g. the arrival rate of complaining customers). By experimenting with such factors a greater understanding of the real system can be obtained. This, after all, is a key benefit of simulation (Robinson, 2004).

Where possible, it is useful to determine the range over which the experimental factors are to be varied. This can be achieved through discussion between the modelling specialist and the clients. In the considered case study, if the number of technical staff on a shift is being investigated, what is the minimum and maximum number possible? The simulation model can then be designed to enable this range of data input. On some occasions this helps to avoid an over-complex model design that provides for a much wider range of data input than is necessary.

There should also be some discussion on the method of data entry for the experimental factors. This might be direct into the model code, through a set of menus, through a data file or via third party software such as a spreadsheet. In large measure this depends upon the intended users of the model and their familiarity with computer software. This decision relates to the general project objectives discussed above.

Similarly, the identification of the responses required from the model should not provide a major challenge. The responses have two purposes. The first is to identify whether the objectives have been achieved. For example, if the objective is to increase throughput of a production site by a certain amount, then it is obvious that the model needs to report the throughput. The second purpose of the responses is to point to the reasons why the objectives are not being achieved. Taking the throughput example, this might require reports on machine and resource utilization and buffer/work-in-progress levels at various points in the model. By inspecting these reports, the user should be able to identify potential bottlenecks, and look for solutions (Robinson, 2004).

Another issue to be considered is how the information is reported, for instance, as numerical data (mean, maximum, minimum, standard deviation) or graphical data (time-series, histograms, Gantt charts, pie charts). The identification of suitable responses and methods of reporting should be determined by close consultation between the simulation specialists and the clients. The nature of the reports depends upon the requirements for visual and interactive features in the model, as outlined in the discussion on general project objectives above.

Table 4.3 shows the relevant experimental factors and responses for the power plant example.

As with all aspects of the modelling process, both the experimental factors and responses will change as the project progresses. It may be realized, for instance, that changing the number of technicians is not effective in improving customer service, but that changing the human resources supervising and management is. As experimentation progresses, the need to inspect reports on the level of rework to understand the restrictions in throughput may become apparent. The experimental factors and responses may also change as a result of changes to the problem situation, the understanding of the problem situation or the modelling objectives (Robinson, 2004).

Table 4.3. Power Plant Illustration – Experimental Factors and Responses

Experimental Factors and Responses
Experimental Factors - Total number of technical staff at each hour of the day **Responses** (to determine achievement of objectives) - Percentage of complaining customers waiting for less than 1 hour **Responses** (to identify reasons for failure to meet objectives) - Histogram of waiting time for each customer, mean, standard deviation, minimum and maximum - Staff utilization (cumulative percentage)

It should be apparent from the description above that the modelling objectives are central to the conceptual modelling framework described here. It is for this reason that determining the modelling objectives is described as part of the conceptual modelling process. Since the understanding of the problem situation is central to the formation of the modelling objectives, it is also considered to be part of the conceptual modelling process.

4.2.4. The conceptual model design: model content

Having identified the model's inputs and outputs, the modelling specialist can identify the content of the model itself. Although this course is about managing the model-based simulation, the need to consider the appropriate modelling approach should not be forgotten at this point. In designing the content of the model, and indeed before this point is reached, the specialist should consider whether simulation is the most suitable approach (also taking into account some alternatives to be used whenever possible).

Assuming that simulation is the answer, the starting point in designing the model content is to recognize that the model must be able to accept the experimental factors and to provide the required responses. The experimental factors and responses provide the basis of what the model needs to include. Taking the example of technical staff shifts, it is immediately obvious that the model must represent these (as numbered lists, for example). The model must then provide the relevant reports, for instance, complaining customers waiting time. It is likely that such a model must include some queues as basic data structures.

Having identified the immediate entry point of the experimental factors, and exit point of the responses, the modelling specialist must then identify the key interconnections between these and the other components of the real world. It is only those interconnections that are judged to be important, with respect to correctly interpreting the experimental factors and providing accurate values for the responses that need to be included in the model. It is probably useful first to think in terms of the scope and then the level of detail (Robinson, 2004).

The scope of the model must be sufficient to provide a link between the experimental factors and the responses. For instance, a model that looks at the throughput (response) resulting from a particular production schedule (experimental factor) needs to include at least all the critical processes within the manufacturing flow from entry of the schedule to creation of the finished items. The scope must also include any processes that interconnect with this flow such that they have a significant impact on the responses, the meaning of significant being defined by the level of model accuracy required. For instance, the manufacturing model must include any processes that interconnect with the production flow and have a significant impact on the throughput. If the supply of raw materials has only a small impact on the throughput, because material shortages are rare, then it is probably unnecessary to model them. If a high level of model accuracy is needed, however, then it is more likely that the supply of raw materials (or at least the shortage of raw materials) needs to be modelled (Robinson, 2004).

The level of detail must be such that it represents the components defined within the scope and their interconnection with the other components of the model with sufficient accuracy. This again can be considered with respect to the impact on the model's responses. For example, considering a single machine on a manufacturing line, the cycle time and breakdowns are very likely to have a significant impact on throughput. Beyond this, the small variations in the machine cycle, the type of machine failure etc., are probably of little importance to accurately predicting throughput, and so can be excluded from the model.

Prototyping is a powerful method in helping to form a decision about the scope and level of detail to include in a model (Powell, 1995; Pidd, 1999). The modelling specialist develops simple computer models, gradually increasing the scope and level of detail. The intention is to throw these models away and not to use them for formal analysis, although they can often provide useful insights for the clients. Their primary purpose is to provide an insight into the key variables and interconnections in order to help with the design of the conceptual model.

In designing the simulation, the modeller must always keep in mind the general project objectives. If the requirement is for a complex visual display, then additional detail may need to be added to the model. If the time-scale is limited, then the scope and level of detail in the model may need to be reduced, possibly compromising on accuracy.

It is also important to keep a record of any assumptions that are made during the design of the model content. They need to be presented to all involved in the simulation study to ensure that everyone understands and agrees the assumptions that are being made. Any simplifications should be noted and explained as well.

Table 4.4 shows the proposed scope of the power plant model, with a justification for what is to be included and excluded. Table 4.5 provides similar information for the level of detail. *These tables represent the conceptual model as a component list*.

Table 4.4. Power Plant Illustration – Model Scope

Model Scope		
Component	**Include/Exclude**	**Justification**
Customers	Include	Flow through the service process
Staff – service	Include	Experimental factor, required for staff utilization response
Staff – supply chain management	Exclude	Supply materials shortage is not significant
Staff – site buildings maintenance	Exclude	Does not interfere with the service
Queues at customer care desk or phone line	Include	Required for waiting time and queue size response

Throughout the development of the conceptual model, the modelling specialist should look for opportunities to simplify the model. This is the subject of the next section.

Table 4.5. Power Plant Illustration: Model Level of Detail

Model Level of Detail			
Component	**Detail**	**Include/Exclude**	**Comment**
Customers	Customer inter-arrival times	Include	Modelled as a distribution
	Type of customer complaint	Exclude	Represented in service time
Service staff	Service time	Include	Modelled as a distribution, taking account of variability in performance and type of customer complaint/system fault
	Number of technicians per shift	Include	Experimental factor
	Absenteeism	Exclude	Not explicitly modelled, but could be represented by perturbations to number of technicians per shift
Queues at customer care desk or phone line	Queuing	Include	Required for waiting time and queue size response
	Capacity	Exclude	Assume no effective limit
	Queue behavior	Exclude (except joining the shortest queue)	Behavior not well understood

4.2.5. The data significance

Preliminary or contextual data are required for developing an understanding of the problem situation and so are central to the development of conceptual modelling. Meanwhile, data for model realization (developing the computer model) are not required for conceptual modelling, but are identified by the conceptual model (Robinson, 2004).

When accurate data for any part of a process can be obtained, the conceptual model may be designed without consideration for whether the data can be gathered. But this happens only theoretically. In reality, not all data are readily available or indeed collectable and sometimes it is impossible to obtain adequate data, making the proposed conceptual model problematic (Robinson, 2004). This leaves the modelling specialist with two options:

- To redesign the conceptual model in such a way as to engineer out the need for troublesome data.
- To resist changing the conceptual model and to handle the data in other ways.

In practice, the specialists probably use a mixture of the two approaches. As such, the conceptual model defines the data that are required, while the data that are available, or collectable, affect the design of the conceptual model. This serves to increase the level of iteration required in the modelling process, as the modeller must move between consideration for the design of the model and the availability of data (Robinson, 2004).

4.2.6. The conceptual modelling framework – a summary

The framework described above consists of four key stages:

- Developing an understanding of the problem situation.

- Determining the modelling objectives.
- Determining the model inputs and outputs.
- Designing the model content.

It is also necessary to consider whether simulation is the most appropriate modelling approach as part of the conceptual modelling process. The aim of the framework is to provide the modelling specialist with some guidance over how to design the conceptual model. Throughout the design process, the specialist must take into account the four requirements of a conceptual model described in the previous chapter: validity, credibility, utility and feasibility, as well as to develop a model that is as simple as possible.

Many iterations of conceptual model design and client interaction are required. It is not a case of providing a design and then going ahead and developing the computer model. Frequent iterations between model coding, experimentation and model design are also necessary. Practically, the conceptual model will change as the simulation study progresses.

4.3. Model Simplification

Apart from having a framework for conceptual modelling, it is also useful for the modeller to have some methods of model simplification. As previously shown, simplifications are not assumptions about the real world, but they are ways of reducing the complexity of a model. Model simplification involves reducing the scope and level of detail in a conceptual model by two means:

- Removing components and interconnections that have little or no effect on model accuracy.
- Representing more simply components and interconnections while maintaining a satisfactory level of model accuracy.

This can either be achieved by identifying opportunities for simplification during conceptual modelling or once the conceptual model is complete and beyond, for instance, during model coding (Robinson, 2004). The main purpose of simplification is to increase the utility of a model while not significantly affecting its validity or credibility. In general, simplification enables more rapid model development and use. Simplification may be necessary if the original model design is infeasible, for instance, because required data are not available.

Before shortly describing some methods of model simplification, it is worth noting that one of the most effective approaches for simplifying a model is, in fact, to start with the simplest model possible and gradually to add to its scope and level of detail (Pidd, 1999). Once a point is reached at which the study's objectives can be addressed by the model, then no further detail should be added (Robinson, 2004). Finding an appropriate point at which to stop, however, requires careful attention on the part of the modelling specialist, but the framework described earlier in this chapter should aid this process.

4.3.1. Model components aggregation

Aggregation of model components provides a means for reducing the level of detail. Two specific approaches are described here: black-box modelling and grouping entities.

Black-box modelling

In black-box modelling a section of an operation is represented as a time delay. Model entities that represent parts, people, information and such like enter the black-box and leave at some later time. This approach can be used for modelling anything from a group of machines/devices to a complete factory or plant.

Figure 4.2 illustrates the approach. As an entity X_i enters the black-box, the time at which it is due to leave, t_i, is calculated. When the simulation reaches time t_i, the entity leaves the box. The time an entity spends in the box can of course be sampled from a distribution. The approach can also be extended to account for re-sequencing of entities (e.g. re-work), stoppages and shifts by manipulating the values of t_i for each entity in the box.

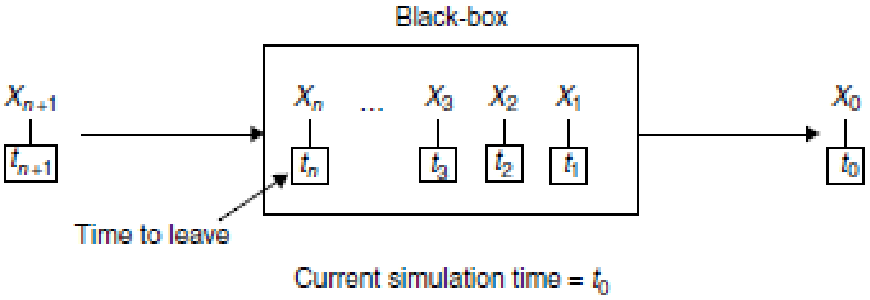


Figure 4.2. Black-Box Modelling.

Grouping entities

Instead of modelling individual items as they move through a system, a simulation entity can represent a group of items. This is particularly useful when there is a high volume of items moving through a system, for example, a confectionery wrapping process in which hundreds of products are wrapped each minute. To model each product individually would lead to hundreds of events per simulated minute, which would have a negative effect on simulation run-speed. It is beneficial in this case for an entity to represent, say, 100 products.

The approach can easily be adapted to model situations where the number of items represented by an entity changes as the entity moves through the model. For example, a certain number of products are rejected at an inspection area. This can be modelled by holding as an attribute of the entity the number of products it represents. The attribute value can then be adjusted as the entity moves through the model.

4.3.2. Excluding components and details

On occasions it is not necessary to include some components in a simulation because their omission has little effect on the accuracy of the model. This is a form of scope reduction.

Resources required for a process to take place need not be modelled if it can be assumed that the resource is always, or almost always, available to perform that task. In this case it is only necessary to model the process. For instance, an operator who is dedicated to a task on a manufacturing line need not be modelled explicitly (Robinson, 2004).

The same author gives an interesting image over this approach: *The modelling of machine repairs provides a very specific example of model simplification, in this case driven by the availability of appropriate data. If the resources required for repair (normally maintenance operators and possibly some equipment) are to be modelled explicitly, then it is necessary to have data on actual repair times. However, many organizations only collect data on machine down times, that is, the total time the machine is down including the time for the resources to be made available. If down time data are being modelled, then the resources should not be explicitly included in the simulation, otherwise a form of double counting is taking place*.

Some details may be excluded from a model because they also have little impact on model accuracy. An example would be the modelling of shift patterns. These only need to be modelled if:

- Different areas work to different shifts.
- The availability of labor, process speed or process rules vary between shifts.
- Operations continue outside of shifts (i.e. machine repair).
- Shifts need to be modelled to give the simulation credibility.

Otherwise, it is unnecessary to model the dead time between shifts.

4.3.3. Using random variables

Rather than modelling a component or group of components in detail it may be possible to represent them as a set of random variables, sampled from some distributions. For instance, modelling transportation systems such as fork-lift trucks, automatic guided vehicles, heavy goods vehicles or trains can be complex. Depending on the context, allowance needs to be made for breakdowns, traffic congestion, weather conditions, turnaround times and driver shifts.

4.3.4. Rare events exclusion

Some events only affect an operational system on an infrequent basis. A power line (for energy distribution) may be completely burned by lightning once every 5 years. It is probably best to exclude the possibility of such events occurring during a simulation run so as to investigate the operational system under normal working conditions. The effect of such events can always be investigated by performing specific runs in which the event is forced on the model.

4.3.5. Reducing the rule set

Rules are used in simulation models to determine routes, processing times, schedules, allocation of resources and so on. A model can be simplified by reducing the rule set, while maintaining a sufficient level of accuracy. In many cases, 80% of circumstances are covered by 20% of the rule set, for instance, routing decisions for automatic guided vehicles. Judgment is required as to whether it is worth modelling the other 80% of the rule set for a small improvement in model accuracy (Robinson, 2004).

One specific difficulty in simulation modelling is to represent the human interaction with an operational system (i.e. it is very difficult to know how people behave when queuing in a service system). In this case it is practically impossible to assume a valid rule set for all people in all situations. Therefore, normal practice is to use a simplified set of rules, for instance, "customers choose the shortest queue" or "they will not join a queue if there are more than five people in it".

An extreme but useful approach is to neglect all rules (Robinson, 2004). In the service system example above the simulation could make no assumptions about queuing behavior except assuming people join the shortest queue. This would mean that if there is an imbalance between service rate and arrival rate the queues would become very large. This gives useful information, that is, the system is not balanced and custom is likely to be lost unless the service rate can be increased.

4.3.6. Splitting models

Instead of building one large model, it can be useful to split the model into parts. A simple way of achieving this is to split the models such that the output of one sub-model (model A) is the input to another (model B), as seen in Figure 4.3. As model A runs, data concerning the output from the model, such as output time and any entity attributes, can be written to a data file. Model B is then run and the data read such that the entities are recreated in model B at the appropriate time.

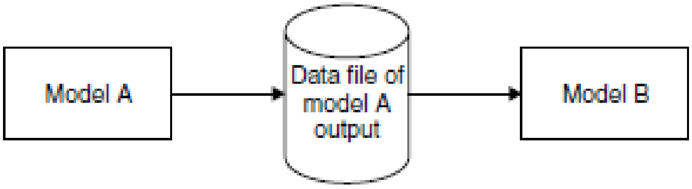


Figure 4.3. Split Models.

The advantage of splitting models is that the individual models run faster. It is also quite probable that a single run of all the sub-models is quicker than one run of a combined model. Another advantage of splitting models is that it is possible to speed development time by having separate modelling specialists for each model in parallel.

But splitting models may not be successful when there is feedback between the models. For instance, if model B cannot receive entities, because the first buffer is full, then it is not possible to stop model A outputting that entity, although in practice this is what would happen. It is best, therefore, to split models at a point where there is minimal feedback (i.e. where there is a large buffer – Robinson, 2004).

There is much interest in running simulations in parallel on separate computers, with the aim of gaining run-speed advantages. If split models run in parallel, then it should be possible to model feedback effects and so overcome the difficulty described above. At present, however, there are a number of obstacles to the use of parallel computing for simulation, the most important being that an efficient mechanism for synchronizing the models as they run still has to be found.

4.3.7. Estimating the simplification quality

Although model simplifications are beneficial, a poor choice of simplification, or oversimplifying a model, may seriously affect the accuracy of the simulation. A good simplification is one that brings the benefits of faster model development and run-speed (utility), while maintaining a sufficient level of accuracy (validity). Two approaches are used in order to determine whether a simplification is good or not.

The first is to use judgment in deciding whether a simplification would have a significant effect on model accuracy. This should be determined by discussion between the modelling specialist, client and other members of the simulation project team. The project specification is a useful mechanism for explaining and discussing the efficacy of proposed simplifications. Of course, this approach provides no certainty over whether a simplification is appropriate or not.

The second approach is to test the simplification in the computer model. The modelling specialist develops two computer models, one with and one without the simplification. It is then possible to compare the results from the two models to see the effect on accuracy. This, of course, provides much greater certainty over the simplification quality, but the advantage of faster model development is lost.

Apart from maintaining a sufficient level of accuracy (validity), a good simplification should not compromise credibility. Over-simplification can make a model less transparent, reducing its credibility. For example, although a black-box may provide a sufficiently accurate representation of part of an operational system, the representation details are not transparent. For some clients this may be satisfactory, but for others it may be necessary to provide a more detailed representation to give the model credibility. It is sometimes necessary to include a greater scope and level of detail than is required to assure the accuracy of the model, in order to assure the model's credibility. A poor simplification is one that causes a client to lose confidence in a model (Robinson, 2004). Indeed, there are occasions when it is necessary to reverse the concept of simplification and actually increase the complexity (scope and level of detail) of the model, simply to satisfy the requirement for credibility.

4.4. Chapter's Conclusions

The issue of how to develop a conceptual model is discussed from two perspectives:

- by presenting a framework for conceptual modelling, enabling a modelling specialist to design a conceptual model from ground;
- by describing a number of methods for simplifying an existing conceptual model.

The framework is illustrated with reference to an example of a power plant. A final issue that has not been discussed is the validation of the conceptual model, but – except some guiding remarks – this is beyond the coverage area of this course.

5.
REFERENCES

Balci, O. and Nance, R.E. (1985) ''Formulated problem verification as an explicit requirement of model credibility''. *Simulation*, 45(2), 76–86.

Banks, J., Carson, J.S. and Nelson, B.L. (1996) *Discrete-Event System Simulation*, 2nd edn. Upper Saddle River, NJ: Prentice-Hall.

Bard, J.F. (1997) ''Benchmarking simulation software for use in modeling postal operations''. *Computers and Industrial Engineering*, 32(3), 607–625.

Brooks, R.J. and Tobias, A.M. (1996) ''Choosing the best model: level of detail, complexity and model performance''. *Mathematical and Computer Modelling*, 24(4), 1–14.

Checkland, P.B. (1981) *Systems Thinking, Systems Practice*. Chichester, UK: Wiley.

Chwif, L., Barretto, M.R.P. and Paul, R.J. (2000) ''On simulation model complexity''. *Proceedings of the 2000 Winter Simulation Conference* (Joines, J.A., Barton, R.R., Kang, K. and Fishwick, P.A., eds). Piscataway, NJ: IEEE: pp. 449–455.

Coyle, R.G. (1996) *System Dynamics Modelling: A Practical Approach*. London: Chapman & Hall.

Dahl, O. and Nygaard, K. (1966) ''SIMULA: an Algol-based simulation language''. *Communications of the ACM*, 9(9), 671–678.

Davis, L. and Williams G. (1994) ''Evaluating and selecting simulation software using the analytic hierarchy process''. *Integrated Manufacturing Systems*, 5(1), 22–32.

Eden, C. and Ackermann, F. (2001) ''SODA–the principles''. In *Rational Analysis for a Problematic World Revisited*, 2nd edn. (Rosenhead, J.V. and Mingers, J., eds), Chichester, UK: Wiley, pp. 21–41.

Fiddy, E., Bright, J.G. and Hurrion, R.D. (1981) ''SEE-WHY: interactive simulation on the screen''. *Proceedings of the Institute of Mechanical Engineers C293/81*, pp. 167–172.

Hlupic, V. and Paul, R.J. (1996) ‘‘Methodological approach to manufacturing simulation software selection’’. *Computer Integrated Manufacturing Systems*, 9(1), 49–55.

Holder, K. (1990) ‘‘Selecting simulation software’’. *OR Insight*, 3(4), 19–24.

Innis, G. and Rexstad, E. (1983) ‘‘Simulation model simplification techniques’’. *Simulation*, 41(1), 7–15.

Law, A.M. (1991) ‘‘Simulation model's level of detail determines effectiveness’’. *Industrial Engineering*, 23(10), 16–18.

Law, A. and McComas, M. (1989) ‘‘Pitfalls to avoid in the simulation of manufacturing systems’’. *Industrial Engineering*, 21(5), 28–69.

Law, A.M. and Kelton, W.D. (2000) *Simulation Modeling and Analysis*, 3rd edn. New York: McGraw-Hill.

Morris, W.T. (1967) ‘‘On the art of modeling’’. *Management Science*, 13(12), B707–717.

Nikoukaran, J. and Paul, R.J. (1999) ‘‘Software selection for simulation in manufacturing: a review’’. *Simulation Practice and Theory*, 7(1), 1–14.

Overstreet, M.C. and Nance, R.E. (1985) ‘‘A specification language to assist in analysis of discrete event simulation models’’. *Communications of the ACM*, 28(2), 190–201.

Pidd, M. (1998) *Computer Simulation in Management Science*, 4th edn. Chichester, UK: Wiley.

Pidd, M. (1999) ‘‘Just modeling through: a rough guide to modeling’’. *Interfaces*, 29(2), 118–132.

Pidd, M. (2003) *Tools for Thinking: Modelling in Management Science*, 2nd edn. Chichester, UK: Wiley.

Pooley, R.J. (1991) ‘‘Towards a standard for hierarchical process oriented discrete event diagrams’’. *Transactions of the Society for Computer Simulation*, 8(1), 1–41.

Powell. S.G. (1995) ‘‘Six key modeling heuristics’’. *Interfaces*, 25(4), 114–125.

Richter, H. and Marz, L. (2000). ‘‘Toward a standard process: the use of UML for designing simulation models’’. *Proceedings of the 2000 Winter Simulation Conference* (Joines, J.A, Barton, R.R., Kang, K. and Fishwick, P.A., eds). Piscataway, NJ: IEEE, pp. 394–398.

Robinson, S. (1994) ‘‘Simulation projects: building the right conceptual model’’. *Industrial Engineering*, 26(9), 34–36.

Robinson, S. (2001) ‘‘Soft with a hard centre: discrete-event simulation in facilitation’’. *Journal of the Operational Research Society*, 52(8), 905–915.

Robinson, S. (2004). Simulation: *The Practice of Model Development and Use*, UK: Wiley.

Robinson, S. and Pidd, M. (1998) ‘‘Provider and customer expectations of successful simulation projects’’. *Journal of the Operational Research Society*, 49(3), 200–209.

Saaty, T.L. (1980) *The Analytic Hierarchy Process*. New York: McGraw-Hill.

Salt, J. (1993) ‘‘Simulation should be easy and fun’’. *Proceedings of the 1993 Winter Simulation Conference* (Evans, G.W., Mollaghasemi, M., Russell, E.C. and Biles, W.E., eds). Piscataway, NJ: IEEE, pp. 1–5.

Schriber, T. (1974) *Simulation Using GPSS*. New York: Wiley.

Senge, P.M. (1990) *The Fifth Discipline: The Art and Practice of the Learning Organization*. London: Random House.

Shannon, R.E. (1975) *Systems Simulation: the Art and Science*. Englewood Cliffs, NJ: Prentice-Hall.

Som, T.K. and Sargent, R.G. (1989) ‘‘A formal development of event graphs as an aid to structured and efficient simulation programs’’. *ORSA Journal on Computing*, 1(2), 107–125.

Sterman, J.D. (2000) *Business Dynamics: Systems Thinking and Modeling for a Complex World*. New York: McGraw-Hill.

Torn, A.A. (1981) ‘‘Simulation graphs: a general tool for modeling simulation designs’’. *Simulation*, 37(6), 187–194.

Ward, S.C. (1989) ‘‘Arguments for constructively simple models’’. *Journal of the Operational Research Society*, 40(2), 141–153.

Willemain, T.R. (1994) ‘‘Insights on modeling from a dozen experts’’. *Operations Research*, 42(2), 213–222.

Zeigler, B.P. (1976) *Theory of Modelling and Simulation*. Chichester, UK: Wiley.

Curs 1

Cap.1. Introducere

1.1.Obiectul cursului

Cursul de *Modelare şi simulare* se ocupă cu studiul principiilor, metodelor şi tehnicilor prin care obiecte din lumea reală, fenomene, operatii si instalatii tehnologice, (numite generic *procese*), sunt *reprezentate matematic* şi apoi *analizate indirect* utilizând tehnica de calcul.
Modelarea si simularea sunt, într-un mod specific, etape esentiale, necesare, in majoritatea activităţilor umane.
Astfel, in general, se parcurg următoarele etape:

- analiza de sistem – implică formularea problemei, precizarea scopurilor, delimitarea dintre „sistemul" studiat şi „mediu" (tot ceea ce este „exterior" sistemului). Se pun în evidenţă mărimile caracteristice, factorii specifici ş.a.m.d.
- modelare - se determină relaţiile dintre mărimile caracteristice, se construieşte o „imagine" a obiectului real , un model „simplificat" al procesului considerat.
- simulare – presupune efectuarea unor „experimente" cu modelul, testarea si validarea modelului , prevederea evoluţiilor viitoare;
- decizie, acţiune - în care pe baza rezultatelor experimentelor de simulare se determină acţiuni , se iau decizii (inclusiv decizii de conducere) etc.

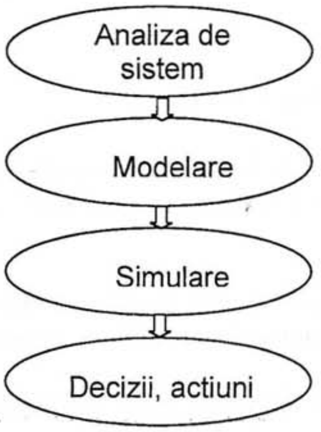


Fig.1. Etape

Indiferent care este scopul unei activităţi umane raţionale, după precizarea si delimitarea problemei, analistul ia in considerare o serie de factori pe care-i consideră importanţi si construieste o imagine proprie, un „model" al procesului respectiv.
Modelul constituie deci, o reprezentare simplificată, aproximativă a realităţii, in care se ignoră in mod voit (sau poate involuntar) o serie de detalii, dar care este considerat satisfăcător in raport cu obiectivul propus. Folosind acest model, analistul incearcă să prevadă, să deducă cum se vor desfăsura fenomenele, realizand astfel un „experiment de simulare" .
In funcţie de rezultate se pot lua decizii (inclusiv decizii de conducere), se stabilesc acţiuni etc.

1.2.Sistem. Stare.Model. Simulare

Sistem.

In general in domeniul tehnic, un **sistem** este definit ca un obiect sau ansamblu de entități, de elemente interconectate, ce interacționează într-un anumit mod pentru a realiza un obiectiv, un scop, cu anumite performanțe.

În particular, în automatică, obiectul din lumea reală, fenomenul, procesul tehnologic, instalația, se numeşte sistem (fizic).

Tot ceea ce nu aparține *sistemului* face parte din lumea exterioară (*mediu*).

Linia de separație dintre *sistem* şi *mediu* pune în evidență mărimile de intrare I – mărimi „cauză" (u) şi mărimile de ieşire E – mărimi „efect" (y), determinate prin cauzalitatea *intrare-ieşire*.

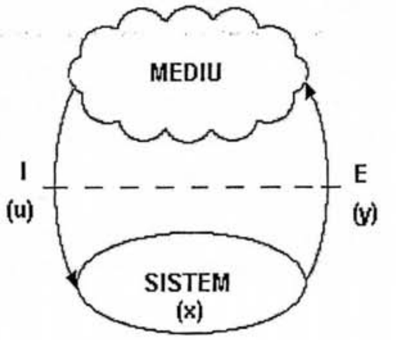


Fig.2. I→ E

Obs. Sistemul depinde esențial de de obiectivele studiului, analizei; ceea ce intr-un caz este considerat un „sistem", in alt context poate fi doar un „subsistem" component al unuia mai complex.

Stare.

Starea x(t) a unui sistem, se defineste ca fiind *informația minimă* necesară la un moment dat de timp *t*, care împreună cu intrările ulterioare *u(t)* , determină univoc evoluția ieşirilor *y(t)*.

Mulțimea tuturor variabilelor de stare (liniar independente) , formează vectorul de stare *x(t)*.

Model.

În domeniul ştiințelor tehnice, experimentul şi observația (măsurarea) constituie aspecte esențiale pentru un model ce se elaborează iterativ.

În ultimă instanță, *elaborarea unei teorii* reprezintă construirea unui *model (verbal* sau *matematic)* al realității.

Def: *Modelul* este reprezentarea într-o formă utilizabilă, a cunoştinţelor, a aspectelor esențiale ale unui sistem.

Observația 1. Modelul este o reprezentare *simplificată,* deci *aproximativă* a sistemului real. Nu e de regulă nici posibil, nici necesar să se realizeze o descriere amănunțită a tuturor mecanismelor interne. E suficient ca modelul să reproducă, să mimeze, suficient de exact comportarea sistemului real.

Observaţia 2. Există multe tipuri de modele şi anume:

- *modele fizice* („empirice" sau „la scară redusă"– de exemplu în domeniul chimiei, elaborarea unei noi tehnologii se incepe cu faza de „micropilot" a instalatiei, cand se testeaza procesul tehnologic pe acest model fizic urmand ca apoi să se realizeze instalaţia industrială).
- *modele fenomenologice* („conceptuale" – sistemele reale respective sunt descrise prin anumite legi fizice)
- *modele funcţionale* („formale" – sistemul e reprezentat prin relaţii funcţionale, scheme funcţionale)
- *modele matematice* („analitice").

Observaţia 3. Modelul trebuie să fie într-o „formă utilizabilă", deoarece *modelul* nu este un scop în sine. El constituie doar o bază pentru analiză, pentru luarea deciziilor; în acest sens modelul trebuie să fie de o complexitate cât mai redusă in concordanţă cu obiectivele studiului.

Simulare.

Definiţie. Simularea este o *metodă experimental-aplicativă* prin care se realizează, se implementează, de obicei pe un calculator, un *model* al unui sistem real în vederea *analizei indirecte* a acestuia.

Modelarea şi simularea sunt instrumente de analiză a sistemului. Simularea este utilă în special în cazurile în care *analiza directă* este imposibilă:

- sistemul nu are încă o existenţă reală (este in faza de proiectare)
- sistemul nu poate fi pus la dispoziţia analistului pentru experimentări directe (ex. in aviatie)
- există pericolul producerii unor pagube prin experimentare directă (ex. baraj hidroenergetic)
- sistemul este caracterizat prin evoluţii foarte lente în timp – (ex. de ordinul lunilor, anilor – cazul sistemelor economice, sociale)
- nu pot fi generate direct condiţiile de experimentare (ex. comportarea dinamică a unei clădiri in cazul cutremurelor).

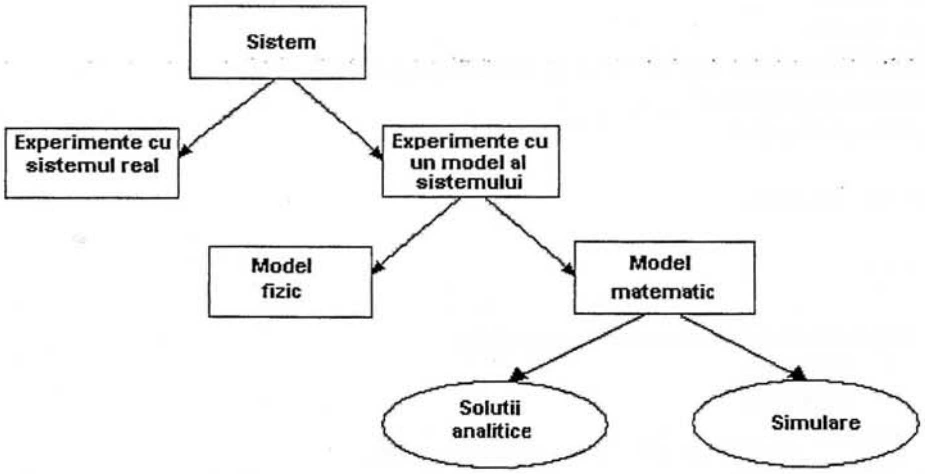

Sistem
Experimente cu sistemul real
Experimente cu un model al sistemului
Model fizic
Model matematic
Solutii analitice
Simulare

Fig.3. De la sistem la simulare

Dezavantajele simulării

- Nu se pot obține soluții, rezultate, foarte exacte, pentru că principial modelele sunt imperfecte (modelele fiind aproximări ale lumii reale, materiale).
- Există erori (în precizarea datelor, a parametrilor, a condițiilor de simulare) care nu pot fi in totalitate compensate.
- În cazul proceselor foarte complexe, *modelul de simulare* poate deveni mai complex decât procesul însuşi.
- Cel mai important dezavantaj este acela că *nu se generează soluții analitice.*

1.3.Clasificarea modelelor

Există o mare diversitate de tipuri, de clase de modele, alegerea modului de reprezentare depinzând de obiectivele studiului. Deasemenea majoritatea criteriilor de clasificare au dezavantajul de a nu reuşi să caracterizeze complet, în totalitate fiecare model în parte.

Criterii

1. După *natura* modelului, există modele:
   - fizice (empirice),
   - fenomenologice (conceptuale),
   - matematice (simbolice -formale sau analitice).

2. După *caracterul dinamic* al modelului, există modele
   - dinamice
   - statice.

În modelele *dinamice* variabilele caracteristice, stările, ieşirile depind şi de „istorie", de evoluția anterioară a acestora.

Ex. model dinamic in reprezentare de stare:

$$\begin{cases} \dot{x} = f(x,u,t) \\ y = g(x,y,t) \end{cases} \quad t \in R$$

Model dinamic in reprezentare de stare in timp discret:

$$\begin{cases} x(k+1) = f(x(k),u(k),k) \\ y(k) = g(x(k),y(k),k) \end{cases}, \text{ unde } k \in Z.$$

Ex. Model static :

$$y = f(u)$$

3.După *gradul de liniaritate* sunt modele:
- liniare şi
- neliniare.

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}, \text{ cu } \Sigma(A,B,C,D) \text{ pentru sisteme } \textit{liniare}.$$

$$\begin{cases} \dot{x} = f(x,u,\theta,t) \\ y = g(x,u,\theta,t) \end{cases}, \text{ - pentru sisteme } \textit{neliniare} \text{ (unde } \theta\text{- este vectorul parametrilor).}$$

Obs. O clasă specială de modele este clasa *modelelor liniare in parametri.*

Ex: $y(k)=\varphi^T(k)\cdot\theta$

unde θ- este vectorul parametrilor iar φ^T este vectorul „observatiilor" intrare-iesire

4.Există modele:

- *variante* în timp sau
- *invariante* în timp.

Ex. model *variant in timp* $\begin{cases}\dot{x}(t)=A(t)x(t)+B(t)u(t)\\ y(t)=C(t)x(t)\end{cases}$

E suficient ca doar una din matricele A, B, C să fie dependentă de timp.

În cazul *modelelor invariante* în timp aplicând aceleaşi intrări decalate cu τ, se obţin răspunsuri identice, dar decalate cu acelaşi interval τ.

$$u(t)\to u(t+\tau)\Rightarrow y(t)\to y(t+\tau)$$

5. După caracterul *structural*, sunt

- modele *funcţionale* (intrare-iesire I/E) şi
- modele *structurale* (intrare-stare-iesire I/S/E).

Ex: Modele *funcţionale* (sau I/E) sunt de tip *funcţie de transfer* (SISO) – in domeniul complex sau din modele in domeniul timp (ec. diferentiale).

Ca *funcţie de transfer*: $Y(s)=H(s)\cdot U(s)$

În domeniul timp: $\sum_{i=0}^{n}a_i\frac{d^iy}{dt^i}=\sum_{i=0}^{m}b_i\frac{d^iu}{dt^i}$ ← model cu „întârziere de ordinul n şi cu anticipare de ordinul m".

Aplicând transformata Laplace se obţine echivalenţa dintre cele doua reprezentari :

$H(s)=\frac{Y(s)}{U(s)}=\frac{B(s)}{A(s)}=\frac{b_0+b_1s+.......+b_m\,s^m}{a_0+a_1s+.......+a_ns^n}$, cu condiţia de „cauzalitate" strictă $n>m$

(neanticiparea iesirii in raport cu intrarea).

Modelele *structurale* sau *I/S/E* (intrare/stare/ieşire): De ex. o forma echivalentă pentru modelele anterioare (pentru $a_n=1$) iar $x(t)$ este starea, poate fi Σ(A,B,C,D) :

$$[\dot{x}]=\begin{bmatrix}0 & 1 & 0...........0\\ 0 & 0 & 1...........0\\ .... & & 1\\ -a_0 & -a_1 & .....-a_{n-1}\end{bmatrix}\cdot[x]+\begin{bmatrix}0\\0\\...\\1\end{bmatrix}\cdot u(t)$$

$$y=[0\ 0.......b_m\ \ b_{m-1}\ ....b_0]\cdot[x]$$

6. Dupa caracterul *continuu* sau *discret* al variabilei independente *timp*:

- modele continue şi
- modele discrete.

- La sistemele discrete reprezentarea se face prin *relaţii de recurenţă* in timp discret (ex. modele de tip ARMAX - AutoRegressive Moving Average with eXogenous)

sau reprezentari polinomiale I/E in operatorul de intarziere q^{-1} unde $q^{-1}f(k)=f(k-1)$, sau echivalent de tip *funcţie de transfer în z*.

Ex. Model ARMAX

$$A(q^{-1})y(k) = B(q^{-1})u(k) + C(q^{-1})e(k)$$
$$A(q^{-1}) = 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na}$$
$$B(q^{-1}) = b_1 q^{-1} + b_2 q^{-2} + \ldots + b_{nb} q^{-nb}$$
$$C(q^{-1}) = 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc}$$
$$Ee(k)e(l) = \lambda^2 \delta_0 (k-l)$$
$$\forall k,l \in \mathrm{Z}$$

La sistemele continue variabilele de intrare u(t) şi de ieşire y(t) sunt funcţii continue în timp.

7. După dependenţa modelului de coordonatele geometrice, modelele se pot împărţi în:
   - modele cu *parametri concentraţi* şi
   - modele cu *parametri distribuiţi.*

La modelele cu parametri concentraţi mărimile variabile au aceeaşi valoare indiferent de coordonatele geometrice ale punctului respectiv în care sunt precizate. In modelele cu parametri distribuiţi, variabilele depind si de coordonatele geometrice , dinamica fiind exprimată prin ecuaţii cu derivate parţiale. De ex. in cazul unui schimbător de căldură de tip *tub in tub*, temperatura produsului $T_p\ (x,t)$ depinde atat de timp cat si de coordonata x in lungul schimbatorului:

$$\frac{\partial T_p(x,t)}{\partial t} = -V_p \frac{\partial T_p}{\partial x} + K_p\left[T_a(x,t) - T_p(x,t)\right]$$

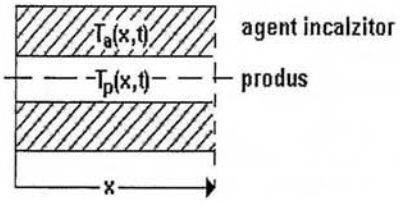


Fig.4. Schimbator de caldura elementar cu parametri concentrati

Obs. Modelele cu parametri distribuiti sunt aproximate deseori in automatica prin modele cu *timp mort.*

8. Deasemenea modelele pot fi
   - *parametrice* sau
   - *neparametrice.*

Modelele parametrice sunt caracterizate printr-un număr *finit* de parametri (ex. coeficientii a_i , b_j din functia de transfer, pe când la modelele *neparametrice* comportarea dinamică se caracterizează prin *reprezentări grafice* în domeniul timp sau frecvenţă.

Ca exemple de modele neparametrice sunt: *răspunsul pondere* (la impuls Dirac), răspunsul *indicial*, *caracteristica de frecvenţă* (Nyquist), *caracteristica Bode*, *caracteristica Nichols.*

9. După numărul de intrari/iesiri (I/O)
   - Modele SISO (Single- Input-Single-Output)
   - Modele MIMO (Multiple- Input-Multiple-Output)
   - Modele SIMO si modele MISO

1.4. Model de simulare. Limbaj de simulare. Experiment de simulare

În cazurile simple, o dată determinat modelul matematic, sistemul poate fi analizat pe baza soluțiilor analitice. În cazul modelelor mai complexe solutia analitica nu mai este posibila, fiind necesară utilizarea unui calculator. Simularea este o metodă experimental – aplicativă prin care se implementează pe un calculator, un model al unui sistem real în vederea analizei indirecte a acestuia. Modelul matematic trebuie insă să aibă o reprezentare adecvată numită *model de simulare.*

Modelul de simulare depinde de programul de simulare sau de simulatorul utilizat. Există o mare diversitate de *programe și medii de simulare*. Actualmente, mediile de simulare specializate, practic, nu mai necesită elaborarea de programe pentru obținerea modelului de simulare, ci prin interfețe grafice utilizator (GUI – Graphical User Interface) asigură posibilitatea selectării prin meniuri a componentelor necesare, realizarea structurii, definirea parametrilor și execuția simulării. Astfel de limbaje și medii de simulare sunt: MATLAB, MATLAB cu SIMULINK, SIMNON, LABVIEW, MATHCAD, MATHEMATICA, MAPLE, SIMAN, VISSIM (Visual Simulation).

Un *experiment de simulare* constă în execuția pe simulator a modelului de simulare pentru seturi de date și/sau parametri specificați. Forma de reprezentare a rezultatelor este foarte diversa , de la cea mai simplă (tabel de valori), până la reprezentări 2D sau 3D și utilizarea animației pentru reprezentarea evoluției în timp.

În particular, simularea este procesul de soluționare, de executie pe calculator a modelelor de tip *schemă bloc.*

Schemele bloc sunt o componentă a unui „mediu de programare grafică "(vizuală) sau „medii de programare orientată pe obiecte"(MPOO).

Cele mai utilizate medii de simulare sunt :

- Matlab – Simulink (firma Mathworks) ;
- Lab View (National Instruments) ;
- VisSim (Visual Solutions) ;
- Easy 5 (Boeing) ;
- Matrixx (Integrated Systems) ;
- MathCad, Simnon, Siman .

Toate mediile de simulare oferă 2 funcții de bază :

1. Editare grafică – pentru crearea, editarea și procesarea modelelor ; editorul poate fi utilizat și pentru crearea modelului intrărilor, simulării (stabilirea condițiilor), prezentarea rezultatelor ;
2. Simulare propriu-zisă – execuția modelului prin iterații succesive – calcul numeric + integrare etc.

Mediile de simulare grafică sunt orientate pe schema bloc, deci nu este efectiv necesară o „programare" propriu-zisă. Cele mai utilizate sunt :

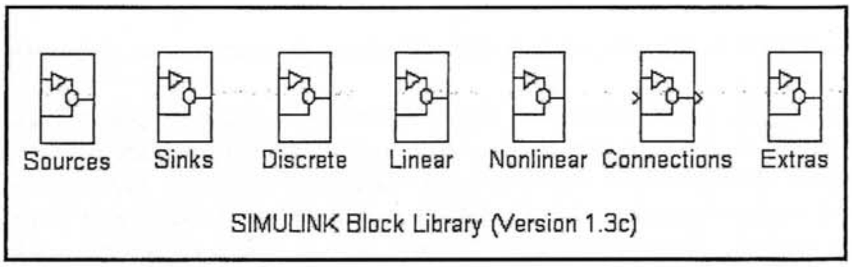
- Matlab – Simulink ;
- VisSim.

Ex. Menu-rile Simulink, blocurile Simulink, crearea unui program.

Etape :

① ›› Simulink

Funcțiile disponibile în Simulink pot fi accesate prin intermediul unor blocuri aflate în biblioteci de funcții Simulink. Fereastra activă permite selecția (double-click) unei subbiblioteci (ex. Simulink v 1.3 c).



Ex.

Linear Library	Sum	Sumator
	Integrator	Integrare semnal
	Gain	Multiplicare cu o constanta a semnalului de intrare
	State-Space	Reprezentarea de stare a unui sistem liniar
	Transfer Fcn	Reprezentarea sub forma de functie de transfer
	Zero-Pole	Reprezentarea sub forma de poli-zerouri

② Se deschide o nouă fereastră pentru crearea modelului.

New...... din meniul File → (se deschide o fereastră „untitled").

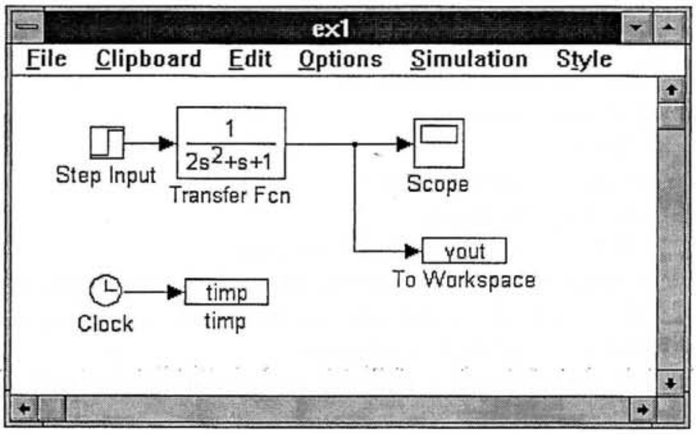
③ Pentru crearea unui model blocurile necesare vor fi „mutate" din subbiblioteci în fereastra activă prin „tragere".

④ Se realizează conexiunile între blocuri – (prin desenare cu mouse-ul apăsat).

⑤ Se configureaza blocurile (se stabilesc parametrii specifici fiecărui bloc)

⑥ Simularea propriu-zisă prin Start din meniul Simulation (anterior fiind stabiliți „parametrii" de simulare – ex: *stop time*).

Ex. de model grafic Simulink



Etapele analizei prin simulare

Simularea este un proces iterativ cu următoarele etape:

1. Stabilirea cadrului simulării - definirea sistemului de analizat, a obiectivelor, a variantelor care se vor avea în vedere, a criteriilor de apreciere;
2. Construirea modelului matematic (modelarea analitică);
3. Realizarea modelului de simulare ;
4. Definirea experimentelor de simulare (inclusiv a datelor pentru validarea modelelor);
5. Experimentul de simulare propriu-zis (verificarea şi validarea modelului, bazata pe experienţa a celui care realizează simularea);
6. Analiza şi interpretarea rezultatelor.

Cursul 2

Cap2. Modelarea analitică a proceselor tehnologice

2.1 Modelare analitică si modelare experimentală

***Modelul** este* o reprezentare sub formă utilizabilă, a cunostintelor, a aspectelor esentiale ale unui sistem.

Modelul matematic este un model exprimat *analitic* prin relatii cantitative specifice (ecuatii diferentiale , ecuatii cu derivate partiale s.a.).

In general un sistem fizic (un obiect din lumea reală) este caracterizat printr-o serie de variabile specifice v=[$v_1,v_2...v_q$] si o serie de relatii $R_i=R[v_1,v_2...v_q]=0$ consecinte a legilor fizice.

In automatică e esentială introducerea unei *orientări* I/E (cauză /efect) in sensul că unele dintre marimile specifice v sunt intrări (mărimi cauză-u), altele sunt iesiri (mărimi efect-y) altele sunt variabile interne (x stări).

$$v=uUyUx$$

astfel incat relatiile R_i vor fi functii de intrări , iesirie, de vectorul de stare x si eventual de vectorii parametrilor θ si timp t, in mod explicit:

$$R_i(u,y,x,\theta,t)=0$$

Se spune că modelul M reprezintă sistemul fizic S, dacă *distanta $D(M,S)$* dintre model si sistem e mai mică decat un ϵ ales corespunzător:

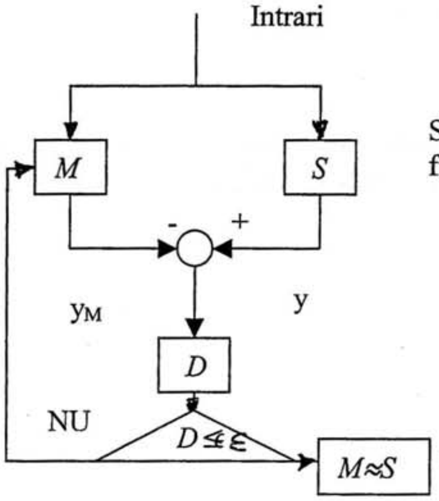
$$D(M,S)\leq \epsilon$$

Exemplu de definire a „distantei" D , pentru un set de N date intrare/iesire:

$$D=\sum_{i=1}^{N}[y(i)\ -y_M(i)]^2$$

$D[M,S]\geq 0$ (semipozitiv definită)

$D[M,S]=0$ => $M=S$ (in realitate este imposibil ca modelul să reflecte „exact" sistemul fizic)



-modelul reprezintă suficient de exact sistemul

Obs1: Modelul matematic reprezintă o *aproximare*, o *simplificare* a realitătii. Modelul *nu poate* (si in general nici *nu trebuie*) să reprezinte *exact* sistemul real in toată complexitatea sa.

Obs2: In acelasi timp modelul matematic are o existentă de sine stătătoare si externă realitătii fizic măsurabile.

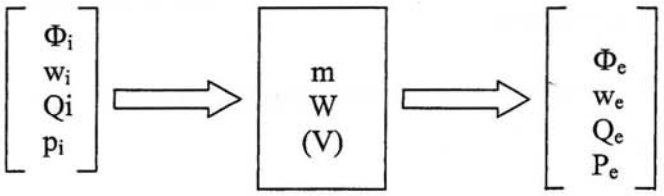
Modelul are un caracter generalizator pentru o clasă de sisteme echivalente indiferent de natura fizică a fenomenelor pe care le caracterizează.

Construirea modelui matematic se poate aborda in două moduri:

1) *modelare analiti*că- ca o consecintă a legilor fizice ce descriu desfăsurarea fenomenelor.
2) *modelare experimentală* (sau *identificare*) in care determinarea modelelor se face prin prelucrarea datelor obtinute din măsuratori experimentale.
   Dacă modelul este cunoscut ca structură, doar parametrii θ fiind necunoscuti, atunci problema determinării modelului se reduce la o problema de "estimare de parametri".

2.2 Etapele modelării analitice

In general procesele tehnologice sunt caracterizate de fluxuri masice sau volumice (Φ/ Q) – numite si debite masice/volumice- si/sau fluxuri de energie- numite si puteri (w/p), care se introduc in proces pentru a fi prelucrate in instalatia tehnologică si a se obtine fluxuri masice si/sau de energie la iesire.



Dacă notăm, m / W, masa / energia, (volumul V) acumulate in process, atunci considerăm că procesul e in *regim stationar* dacă există un echilibru :

$$\Phi_i-\Phi_e=0$$

Procesul este in *regim dinamic* dacă cele două fluxuri nu sunt egale, diferenta lor fiind de fapt egală cu viteza de variatie (de acumulare/evacuare) a masei/energiei (m/W) sau cu variatia acestora in unitatea de timp:

$$\Phi_i-\Phi_e=\frac{dW}{dt}$$

Fluxurile au ca unitate de masură :

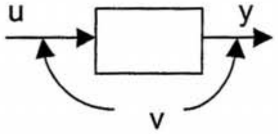
-pentru fluxuri masice (debit masic) $<\frac{kg}{s}>$

-pentru fluxuri volumetrice (debit volumic) $<\frac{m^3}{s}>$

-pentru fluxuri energetice (puteri) $<\frac{J}{s}>$

Indiferent care ar fi in particular procesul al cărui model dorim să-l obtinem , in modelarea analitică se parcurg următoarele etape:

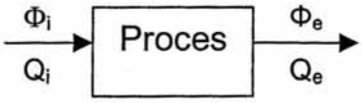
1) evidentierea variabilelor si mărimilor caracteristice : $v_1, v_2, ..., v_q$
2) determinarea pe baza legilor fizice (conservarea energiei, a masei etc) a relatiilor $R_i[v_1, v_2, ..., v_q]$=o intre variabilele caracteristice.
   Tot in aceasta etapă se evidentiază relatiile de *regim stationar* si *regim dinamic*
3) „Orientarea intrare-iesire" a modelului prin punerea in evidenta a variabilelor de tip *cauză (*intrări - *u) si* respectiv *efect*(iesiri - y)



4) „Liniarizarea" modelului presupune :
   -"*centrarea"* variabilelor (trecerea la mici variatii in jurul "punctului static de functionare" PSF, sau trecerea la *variabile centrate*);
   $\Delta v = v - v_0$
   - „*normarea"* variabilelor raportand variabilele centrate la anumite valori de regim stationar (de ex. Valorile corespunzatoare PSF –ului):
   $\frac{\Delta v}{v_0} = v^*$
5) etapa de „validare" a modelului.

2.3 Modelarea proceselor cu acumulare – evacuare de fluid

Procesele se incadrează in clasa de sisteme numite *monocapacitive* (procesele au o singură capacitate care poate acumula masă si/sau energie) fiind reprezentabile prin modele matematice de tip ecuatii diferentiale de ordin I.



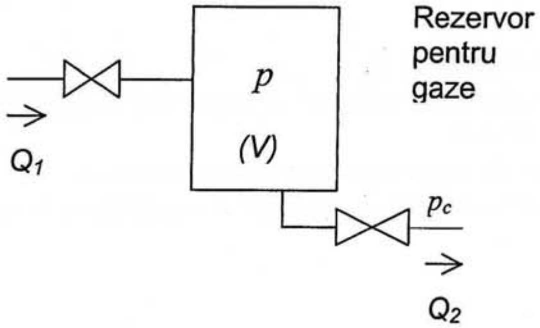
$$\begin{bmatrix} debit\ masic/volumic \\ de\mathrm{int}\,rare \end{bmatrix} - \begin{bmatrix} debit\,masic/volumic \\ de\ iesire \end{bmatrix} = \begin{bmatrix} viteza \\ de\ acumulare \end{bmatrix} = \frac{d\begin{bmatrix} masa/ \\ energie \end{bmatrix}}{dt} = C\frac{dy}{dt}$$

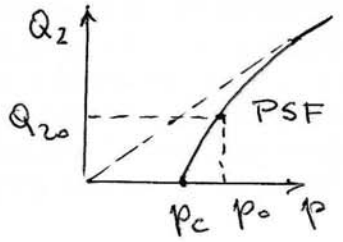
Unde s-a notat in general:

y-variabila dependentă (presiune, nivel, temperatură...)

C-capacitatea elementului de a acumula masă, energie (de ex. energie termică -căldură)

a) Modelarea unui proces de umplere /golire cu gaz





$$Q_2 \cong a\sqrt{p(p-p_c)}$$

Parcurgem etapele standard de modelare analitică:

- Evidentierea variabilelor specifice (masa, densitatea, presiunea in rezervor si cea externa, debitele volumetrice, volumul rezervorului pentru gaze, temperatura absoluta T_k, masa molara a gazului μ ....)
  $v=[m, \rho, p, p_c, Q_1, Q_2, V, \ldots]$
- Legea conservarii masei revine la:
  $$\rho Q_1 - \rho Q_2 = \frac{dm}{dt}$$

-Dar legea gazelor perfecte exprimă dependenta intre *m*- masa de gaz si *p* -presiunea

$$pV = \frac{m}{\mu} RT_k \Rightarrow m = \frac{\mu V}{RT_k} p$$

$$\frac{\mu V}{\rho RT_k} \frac{dp}{dt} = Q_1 - Q_2 \Rightarrow$$

Obs. Se noteaza $C = \frac{\mu V}{\rho RT_k}$ – „capacitatea” pt. gaze a rezervorului

- evidentierea „cauzalitătii” .

Obs. Modelul matematic este neliniar deoarece

$Q_2(p) \cong a\sqrt{p(p-p_c)}$ =>evidentierea cauzalitătii (separarea variabilelor I/E)

$C\frac{dp}{dt} + a\sqrt{p(p-p_c)} = Q_1(t) \Rightarrow$ model matematic neliniar

- liniarizarea modelului:

Dezv. In serie Taylor

$$Q_2(p) = Q_{20} + \left(\frac{\partial Q_2}{\partial p}\right)_0 \cdot (p - p_0) + \left(\frac{\partial^2 Q_2}{\partial p^2}\right)_0 \cdot \frac{(p - p_0)^2}{2!} + \ldots$$

=>introducem „variabilele centrate" punand in evidentă regimul stationar (PSF)

- $\Delta p = p - p_0$
- $Q_2 = Q_{20} + \Delta Q_2$
- $Q_1 = Q_{10} + \Delta Q_1$

Regimul stationar este regimul in care $Q_{10} = Q_{20} = Q_0 = constant$ căruia ii corespunde $p = p_0$. Inlocuind si oprind din dezvoltarea Taylor doar termenul liniar rezultă:

$$C\frac{d(p_0 + \Delta p)}{dt} = Q_{10} + \Delta Q_1 - [Q_{20} + \left(\frac{\partial Q_2}{\partial p}\right)\Delta p]$$

Dacă se tine cont de regimul stationar ($Q_{10} = Q_{20} = Q_0$) si se notează : $\left(\frac{\partial Q_2}{\partial p}\right)_0 = \frac{1}{R_P}$.

inversul unei rezistente pneumatice, se obtine modelul matematic liniarizat in care variabilele I/E sunt „separate"(modelul in variabile „centrate" este „orientat" I/E):

$$C\frac{d\Delta p}{dt} + \frac{1}{R_P}\Delta p = \Delta Q_1$$

Modelul se poate exprima in forma cu „constante de timp" =>

$$R_p C\frac{\Delta p}{dt} + \Delta p = R_p \Delta Q_2$$

unde:

$T = R_p C$ -constanta de timp $[T]$= s - secunde

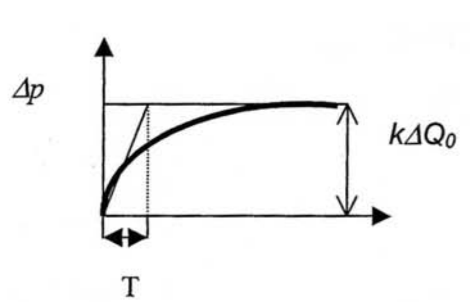
$K = R_p$ -factor de proportionalitate (in regim stationar) [K]= bar/m^3 /s

$$T\frac{d\Delta p}{dt} + \Delta p = K\Delta Q_1$$

Echivalent modelul poate fi reprezentat sub forma „neparametrică" dacă se caracterizează prin „răspunsul" obtinut pentru semnal „treaptă" de debit de intrare, numit „răspuns indicial":

$$\Delta Q_1 = \Delta Q_0$$

=> rezultă răsunsul $\Delta p(t) = K\Delta Q_0[1 - e^{-t/T}]$



Obs. O altă formă echivalentă de reprezentare a modelului este ca *model functional* de tip *functie de transfer*. Se obtine: $H(s)=\dfrac{\Delta p(s)}{\Delta Q_1(s)}=\dfrac{K}{Ts+1}$

Modelul in variabile „normate" poate fi dedus in mod similar introducand:

$$\frac{\Delta p}{p_0}=p^*=y; \qquad \frac{\Delta Q_1}{Q_0}=Q^*=u$$

Se obtine: $$Cp_0\frac{dy}{dt}+\frac{1}{R_p}yp_0=Q_0u$$

In forma echivalentă „cu constante de timp" (inmultim totul cu $\dfrac{p_0}{R_p}$) =>

$$R_pC\cdot\frac{dy}{dt}+y=\frac{R_p}{p_0}Q_0u \quad , \quad \text{sau} \quad T\frac{dy}{dt}+y=K^*u$$

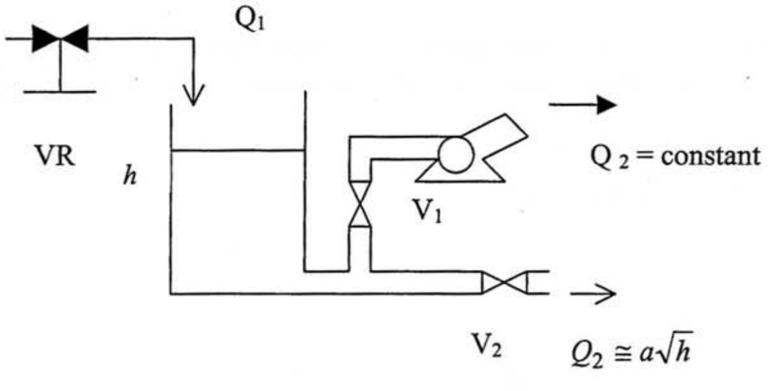
Obs. - Factorul de proportionalitate este evident in acest caz adimensional: $K^{*}=\dfrac{Q_0}{p_0}R_p$

iar constanta de timp este nemodificată $T=R_pC$

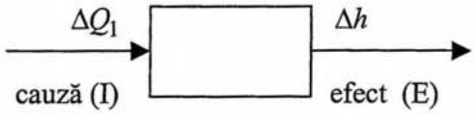
- Daca se calculeaza $y(T)=K^*1[1-e^{-1}]=0{,}63K^* = 63\%\, y_{stationar}$ =>rezultă o metodă simplă de determinare a constantei de timp T
- pentru $R_p\cong R$ statica => $K^*\cong 1$
- u=1(treapta „unitară") este de fapt o variatie a debitului de intrare=> $\Delta Q_1=\Delta Q_0$

b) Modelul proceselor de umplere/golire a rezervoarelor hidraulice

Procesul este un rezervor hidraulic cu A=aria rezervorului constantă si evacuare prin pompa cu debit constant Q_2=*const.* (ventilul V$_2$ - inchis si V$_1$ -deschis) sau prin „cădere liberă" $Q_2\cong a\sqrt{h}$.



Se doreste determinarea unui model matematic care să evidentieze comportarea din punct de vedere al variatiei nivelului de lichid in rezervor (mărime de iesire - „efect"-) atunci cand se modifică debitul de la intrare (mărime de intrare – „cauză")



Se parcurg aceleasi etape:

- Evidentierea variabilelor specifice, $v=[Q_1,Q_2,h,V,\ldots]$
- Legea conservării volumului de fluid (Q – debite volumetrice)

  $$Q_1(t)\text{-}Q_2(t)=\frac{dV}{dt}$$

  $V=f(h)=Ah$=> dacă A= constant, atunci rezultă modelul matematic neliniar:

  $$\mathrm{A}\frac{dh}{dt}=Q_1(t)-Q_2(p,t)$$

- Liniarizarea modelului

In functie de cele două regimuri de functionare ale rezervorului , există două situatii:

b.1) Evacuare prin pompa cu debit constant $Q_2=Q_{20}$ -ventilul de trecere V$_2$- inchis –

Se introduc variabilele centrate $\Delta h=h\text{-}h_0$ unde h_0 corespunde unui PSF (punct static de functionare) in care : $Q_{10}=Q_{20}=Q_0=ct$ si $Q_1(t)=Q_0+\Delta Q_1$

Rezultă astfel:

$$A\frac{d(h_0+\Delta h)}{dt}=Q_0+\Delta Q-Q_0$$

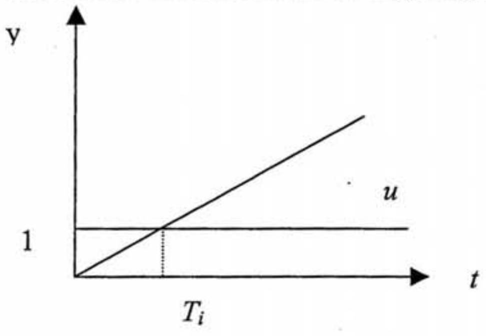
Eliminand regimul stationar => $A\dfrac{d\Delta h}{dt}=\Delta Q_1$

Se trece la variabile normate notand : $y=\dfrac{\Delta h}{h_0}$; $u=\dfrac{\Delta Q_1}{Q_0}$ si avem

$Ah_0\dfrac{\Delta y}{dt}=Q_0 u$ unde $Ah_0=V_0$ astfel

$$\frac{dy}{dt}=\frac{Q_0}{V_0}u. \Rightarrow y(t)=\frac{Q_0}{V_0}\int udt=\frac{1}{T_i}\int udt$$

Acesta este modelul unui sistem „integrator" avand urmatorul „răspuns indicial":

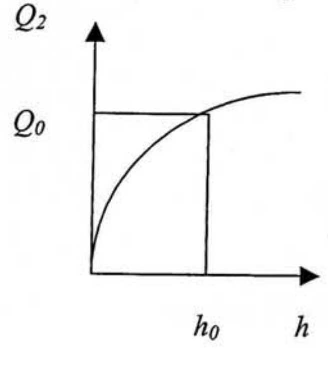


$T_i=\dfrac{V_0}{Q_0}$ -*„constanta de timp de integrare"*

Acest proces e un proces „fără autostabilizare". Echivalentul in reprezentare prin functie de transfer este:

$$H(s)=\frac{Y(s)}{U(s)}=\frac{1}{T_i s}$$

b.2) Evacuare prin „cădere liberă" -ventilul de trecere V$_1$- inchis => $Q_2=Q_2(h)$



Dependenta neliniară (Bernoulli), se liniarizează prin dezvoltare in serie Taylor in jurul PSF (h_0, Q_0)

$$\mathrm{Q}_2(\mathrm{h})=\mathrm{Q}_{20}+\left(\frac{\partial Q_2}{\partial h}\right)_0(h-h_0)+\left(\frac{\partial^2 Q_2}{\partial h^2}\right)_0\left(\frac{(h-h_0)^2}{2!}\right)+\ldots$$

Se trece la variabile centrate :

$\Delta h=h-h_0$

$\Delta Q_1=Q_1-Q_{10}$

$\Delta Q_2=Q_2-Q_{20}$

Modelul liniarizat se obtine prin inlocuire tinand cont că :

$$\left(\frac{\partial Q_2}{\partial h}\right)_0=\left(-\frac{a}{2\sqrt{h}}\right)_0=\frac{a}{2\sqrt{h_0}}$$

Rezultă:

$$A\frac{d(h_0+\Delta h)}{dt}=(Q_{10}+\Delta Q_1)-[Q_{20}+\frac{a}{2\sqrt{h_0}}\Delta h]$$

Se orientează I/E modelul prin separarea variabilelor pentru a evidentia cauzalitatea si se obtine:

$$A\frac{d\Delta h}{dt}+\frac{a}{2\sqrt{h_0}}\Delta h=\Delta Q_1$$

Se trece la variabile normate, notand: $\frac{\Delta h}{h_0}=h^*=y$ si $\frac{\Delta Q_1}{Q_0}=Q_1^*=u$

Rezultă in final modelul liniarizat:

$$Ah_0\frac{dy}{dt}+\frac{a}{2\sqrt{h_0}}h_0 y=Q_0 u \quad \rightarrow \quad \frac{2V_0}{Q_0}\frac{dy}{dt}+y=2u \quad \text{unde } Q_0=a\sqrt{h_0}$$

Parametrii modelului sunt:

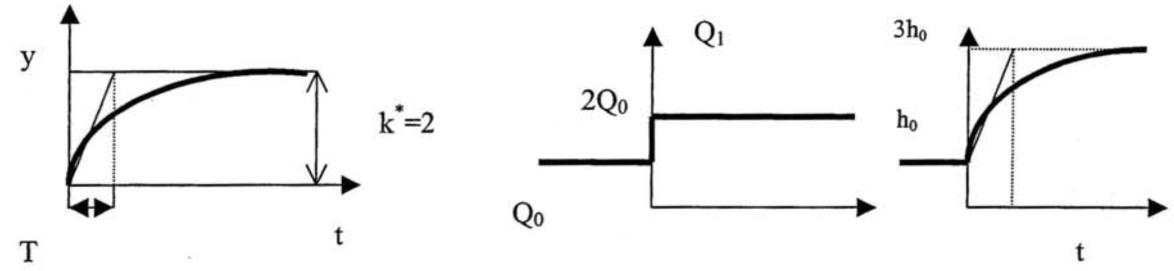
- constanta de timp este $T=\frac{2V_0}{Q_0}$ (dublul timpului de golire) si

- factorul de proportionalitate $K^*=2$ (adimensional)

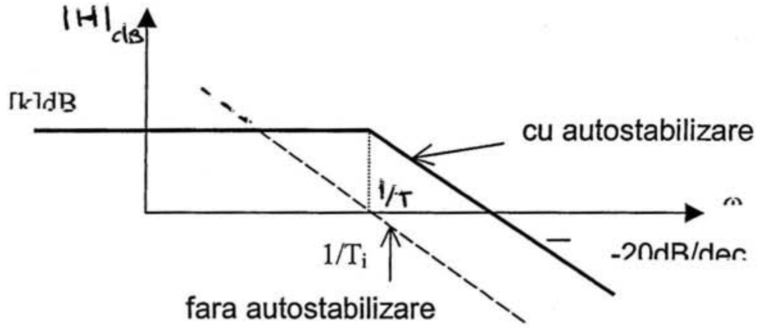
O reprezentare echivalenta a modelului este prin *functia de transfer* (proportional cu intarziere de ordin 1 P-T1):

$H(s)=\frac{Y(s)}{U(s)}=\frac{2}{Ts+1}$ - are un singur pol real egal cu $-\frac{1}{T}$

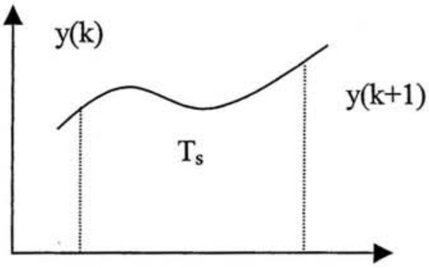
Răspunsul indicial este următorul:



Sistemul se poate deasemenea reprezenta echivalent printr-un alt model „neparametric" si anume *caracteristica de frecventă (caracteristica Bode)*.



Reprezentarea echivalentă ca „model discret" se poate obtine simplu prin „discretizare" aproximand derivata din modelul continuu si trecand la „timpul discret" k=t/T$_s$:



$$T\frac{y(k+1)-y(k)}{T_S}+y(k)=Ku(k)$$

$$y(k+1)=\left(1-\frac{T_S}{T}\right)y(k)+\frac{KT_S}{T}u(k)$$

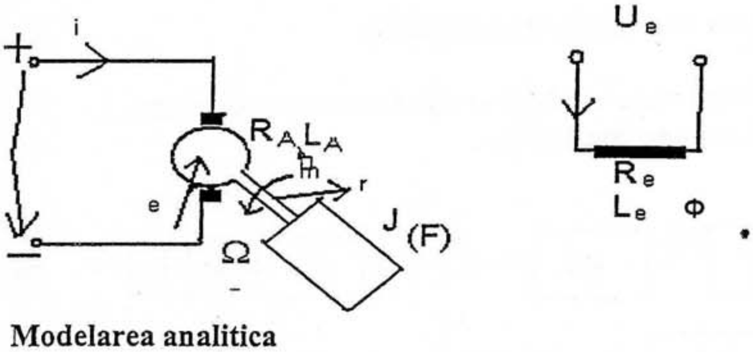
Rezultă modelul discret functional in domeniul timp (relatia de recurentă I/E) sub forma:
$y(k+1)=ay(k)+bu(k)$

Echivalent se poate obtine modelul discret functional in domeniul complex, aplicand transformata Z:
$zY(z)=aY(z)+bY(z)$
Se obtine:

$H(z)=\dfrac{Y(z)}{U(z)}=\dfrac{b}{z-a}=\dfrac{bz^{-1}}{1-az^{-1}}$ reprezentare numită *functia de transfer in z*:

2.8. Modelarea proceselor de conversie electromecanică

Elementul component principal al sistemelor de actionare electrica este motorul electric care constitue *convertor electromecanic* al sistemului de acționare. Se estimează ca aproximativ 40-60% din energia electrică se convertește în energie mecanică prin siteme de actionare electrica. Exemplu: modelul unui motor de curent continuu:



- **Modelarea analitica**

Au loc două fenomene:

1) apariția tensiunii electromotoare *e* prin fenomenul de inductie electromagnetica Tensiunea electromotoare indusă este proporțională cu fluxul de excitație și viteza cu care se rotește rotorul

$$e=K\Phi\,\Omega$$

2) Generarea cuplului electromagnetic

$$m=K\Phi\,i$$

Din echilibrul dinamic intre cuplul mecanic la axul rotoric si cuplul de sarcina rezultă mișcarea de rotație: viteza unghiulara Ω, poziția unghiulara θ. Presupunem că sarcina este caracterizată de momentul de inerție J și de coeficientul de frecare vâscoasă F.

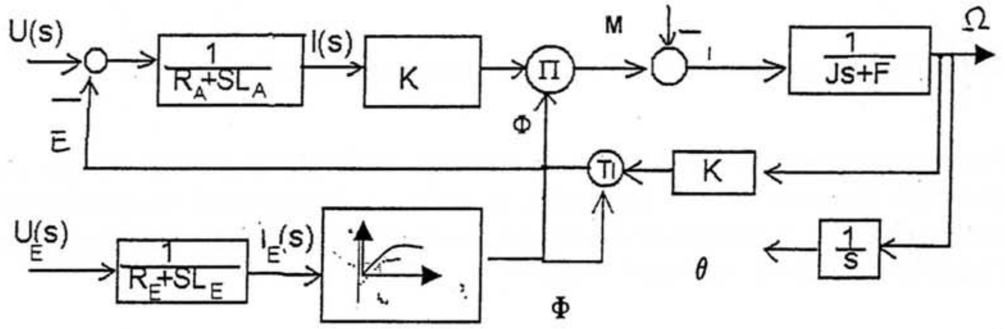
- Scrierea legilor fizice:

Teorema a doua a lui Kirchhoff , relatiile specifice motorului , ecuatia de miscare mecanica precum si ecuatia circuitului de excitatie conduc la:

$$\begin{cases} u(t) = R_A i(t) + L_A \dfrac{di}{dt} + e(t) \\ e(t) = K\Phi(t)\Omega(t) \\ m(t) = K\Phi(t)i(t) \\ J\dfrac{d\Omega}{dt} + F\Omega = m - m_r \end{cases}$$

$$U_E = R_E i_E + L_E \frac{di_E}{dt} \qquad\qquad \Phi = \Phi(i_E)$$

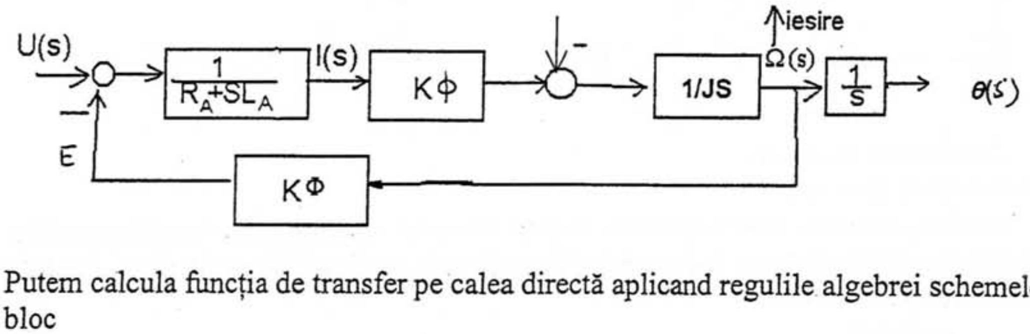
Pentru a obtine o reprezentare ca model funcțional (de tip funcție de transfer) – pentru relațiile liniare se poate aplica transformata Laplace.
Rezulta schema bloc funcțională:



Modelul neliniar poate fi folosit doar în simulare numerica .

Liniarizarea modelului:

1. Dacă Φ este constant, atunci modelul se liniarizeaza in mod natural
Schema bloc simplificată este următoarea:



Putem calcula funcția de transfer pe calea directă aplicand regulile algebrei schemelor bloc

$$H_1 = \left.\frac{\Omega(s)}{U(s)}\right|_{M_r=0} = \frac{\dfrac{1}{R_A+sL_A}K\Phi\dfrac{1}{JS}}{1+\dfrac{1}{R_A+sL_A}K^2\Phi^2\dfrac{1}{JS}} = \frac{\dfrac{1}{(\frac{L_A}{R_A}s+1)}\dfrac{1}{\frac{JR_A}{K^2\Phi^2}S}}{1+\dfrac{1}{(\frac{L_A}{R_A}s+1)}\dfrac{1}{\frac{JR_A}{K^2\Phi^2}S}} =$$

$$=\frac{\frac{1}{K\Phi}\frac{1}{(T_AS+1)T_mS}}{\frac{T_AT_mS^2+T_mS+1}{(T_AS+1)T_mS}}=\frac{\frac{1}{K\Phi}}{T_AT_mS^2+T_mS+1}\approx\frac{\frac{1}{K\Phi}}{(T_mS+1)(T_AS+1)}$$

Se observă că modelul este de tip P-T2 proportional cu întârziere de ordinul 2 (cu poli reali) .

Pe canalul tensiune –curent rezulta similar functia de transfer:

$$H_2(s)=\frac{I(s)}{U(s)}=\frac{\frac{\frac{1}{R_A}}{T_AS+1}}{1+\frac{1}{T_AS+1}\frac{1}{T_mS}}=\frac{\frac{1}{R_A}ST_m}{T_mT_AS^2+T_mS+1}$$ - element de ordinul I cu un zerou

Putem calcula regimul stationar $I_{stationar}=\lim_{s\to 0}s\frac{\frac{1}{R_A}SI_m}{T_mT_sS^2+T_mS+1}\frac{U_0}{s}\to 0$ (pt. Mr =0)

- **Modelul structural**

$$\begin{cases}\frac{di}{dt}=-\frac{R_A}{L_A}i-\frac{K}{L_A}\Omega+\frac{1}{L_A}u\\ \frac{d\Omega}{dt}=\frac{1}{J}Ki-\frac{F}{J}\Omega-\frac{1}{J}m_r\end{cases},$$ unde m_r este perturbația de cuplu

Eventual se poate adăga şi:

$\frac{d\Theta}{dt}=\Omega$ Vectorul de stare fiind $x=\begin{bmatrix}i\\ \Omega\end{bmatrix}$ rezulta:

$$\begin{cases}\begin{bmatrix}\frac{di}{dt}\\ \frac{d\Omega}{dt}\end{bmatrix}=\begin{bmatrix}-\frac{R_A}{L_A} & -\frac{K}{L_A}\\ \frac{K}{J} & -\frac{F}{y}\end{bmatrix}\begin{bmatrix}i\\ \Omega\end{bmatrix}+\begin{bmatrix}\frac{1}{L_A}\\ 0\end{bmatrix}u+\begin{bmatrix}0\\ -\frac{1}{J}\end{bmatrix}m_r\ ;\ =>\begin{cases}\dot{x}=Ax+bu+b_1m_r\\ y=c_1^Tx\end{cases}\\ \Omega=\begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}i\\ \Omega\end{bmatrix}\end{cases}$$

Presupunând cuplul rezistent nul, se pot obține prin calcul aceleasi funcții de transfer:

$$H_1(s)=\frac{\Omega(s)}{U(s)}=c_1^T[sI-A]^{-1}b=\frac{\frac{1}{K\Phi}}{T_mT_As^2+T_ms+1}$$

$$H_2(s)=[1\quad 0][sI-A]^{-1}b.$$

Simulink®

Getting Started Guide

MATLAB®&SIMULINK®

R2019a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Simulink® Getting Started Guide

Trademarks

Patents

Revision History

September 2005	Online only	New for Version 6.3 (Release 14SP3)
March 2006	Online only	Revised for Simulink 6.4 (Release 2006a)
September 2006	Online only	Revised for Simulink 6.5 (Release 2006b)
March 2007	First printing	Revised for Simulink 6.6 (Release 2007a)
September 2007	Second printing	Revised for Simulink 7.0 (Release 2007b)
March 2008	Third printing	Revised for Simulink 7.1 (Release 2008a)
October 2008	Fourth printing	Revised for Simulink 7.2 (Release 2008b)
March 2009	Fifth printing	Revised for Simulink 7.3 (Release 2009a)
September 2009	Online only	Revised for Simulink 7.4 (Release 2009b)
March 2010	Online only	Revised for Simulink 7.5 (Release 2010a)
September 2010	Online only	Revised for Simulink 7.6 (Release 2010b)
April 2011	Online only	Revised for Simulink 7.7 (Release 2011a)
September 2011	Sixth printing	Revised for Simulink 7.8 (Release 2011b)
March 2012	Seventh printing	Revised for Simulink 7.9 (Release 2012a)
September 2012	Eighth printing	Revised for Simulink 8.0 (Release 2012b)
March 2013	Ninth printing	Revised for Simulink 8.1 (Release 2013a)
September 2013	Tenth printing	Revised for Simulink 8.2 (Release 2013b)
March 2014	Eleventh printing	Revised for Simulink 8.3 (Release 2014a)
October 2014	Twelfth printing	Revised for Simulink 8.4 (Release 2014b)
March 2015	Thirteenth printing	Revised for Simulink 8.5 (Release 2015a)
September 2015	Fourteenth printing	Revised for Simulink 8.6 (Release 2015b)
October 2015	Online only	Rereleased for Simulink 8.5.1 (Release 2015aSP1)
March 2016	Fifteenth printing	Revised for Simulink 8.7 (Release 2016a)
September 2016	Sixteenth printing	Revised for Simulink 8.8 (Release 2016b)
March 2017	Seventeenth printing	Revised for Simulink 8.9 (Release 2017a)
September 2017	Eighteenth printing	Revised for Simulink 9.0 (Release 2017b)
March 2018	Nineteenth printing	Revised for Simulink 9.1 (Release 2018a)
September 2018	Twentieth printing	Revised for Simulink 9.2 (Release 2018b)
March 2019	Online only	Revised for Simulink 9.3 (Release 2019a)

Contents

1 Introduction

2 Modeling in Simulink

3 Simple Simulink Model

4 Navigate a Simulink Model

Introduction

Simulink Product Description

Simulation and Model-Based Design

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

Key Features

- Graphical editor for building and managing hierarchical block diagrams
- Libraries of predefined blocks for modeling continuous-time and discrete-time systems
- Simulation engine with fixed-step and variable-step ODE solvers
- Scopes and data displays for viewing simulation results
- Project and data management tools for managing model files and data
- Model analysis tools for refining model architecture and increasing simulation speed
- MATLAB Function block for importing MATLAB algorithms into models
- Legacy Code Tool for importing C and C++ code into models

Model-Based Design with Simulink

Modeling is a way to create a virtual representation of a real-world system. You can simulate this virtual representation under a wide range of conditions to see how it behaves.

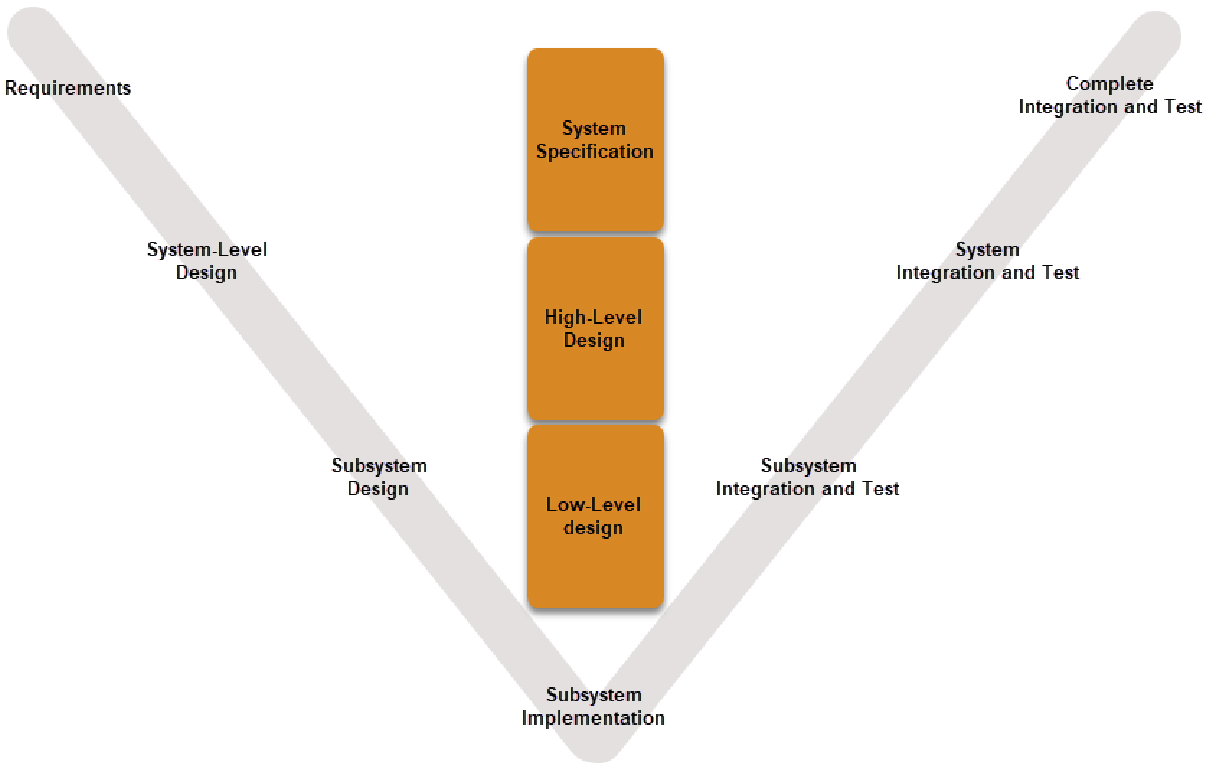
Modeling and simulation are especially valuable for testing conditions that are difficult to reproduce with hardware prototypes alone. This is especially true in the early phase of the design process when hardware is not yet available. Iterating between modeling and simulation can improve the quality of the system design early, by reducing the number of errors found later in the design process.

You can automatically generate code from a model and, when software and hardware implementation requirements are included, create test benches for system verification. Code generation saves time and prevents the introduction of manually coded errors.

In Model-Based Design, a system model is at the center of the workflow. Model-Based Design enables fast and cost-effective development of dynamic systems, including control systems, signal processing systems, and communications systems.

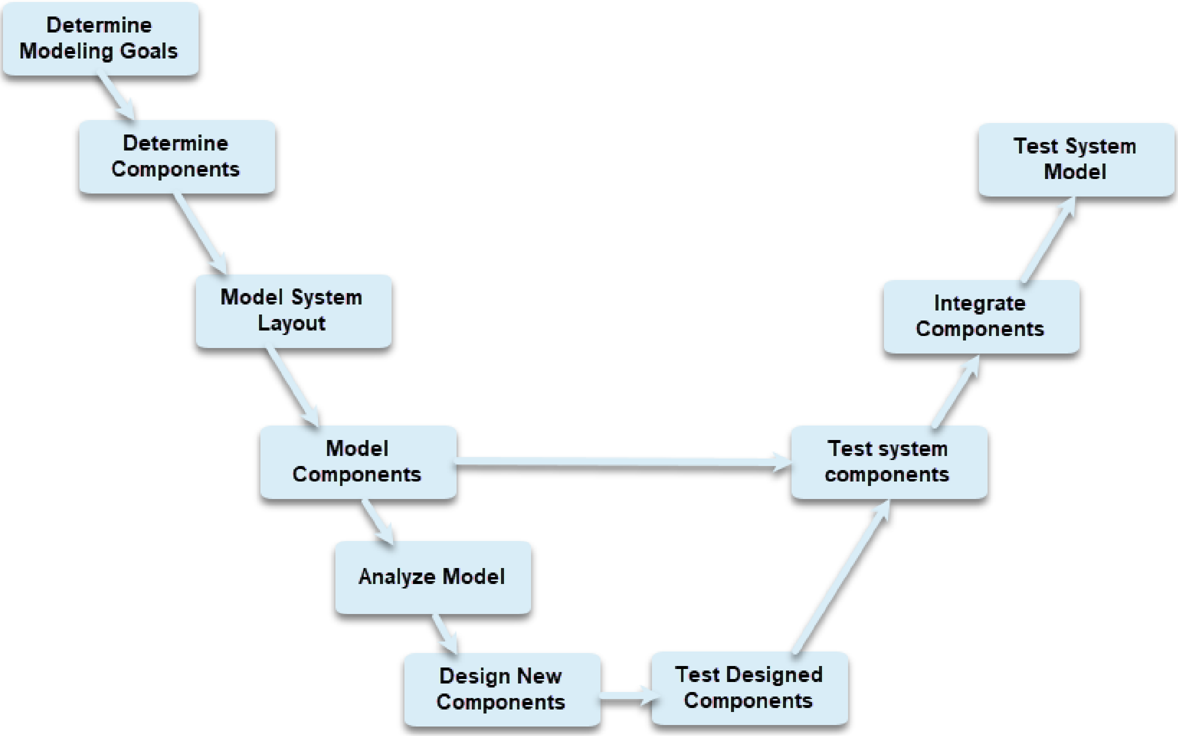
Model-Based Design allows you to:

- Use a common design environment across project teams
- Link designs directly to requirements
- Identify and correct errors continuously by integrating testing with design
- Refine algorithms through multidomain simulation
- Automatically generate embedded software code and documentation
- Develop and reuse test suites

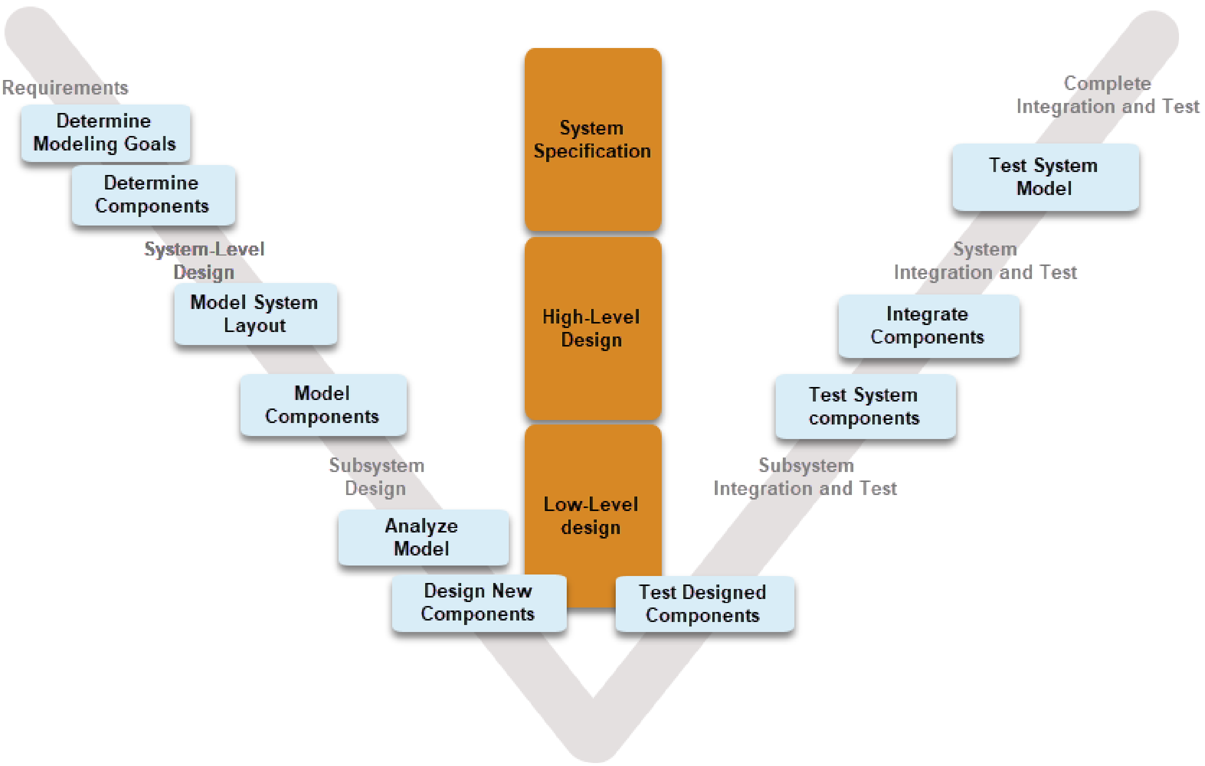


Example Model-Based Design Workflow in Simulink

To get started with a Model-Based Design task, consider this workflow:



The workflow in this tutorial focuses on fundamental Simulink tasks as they relate to Model-Based Design.



For an example workflow, see:

- “System Definition and Layout” on page 1-8 — Identify modeling goals, determine components, model system layout
- “Model and Validate a System” on page 1-16 — Model and test components, integrate components, test system
- “Design a System in Simulink” on page 1-29 — Design and test new components

The first two tasks in this workflow model an existing system and establish the context for designing a component. The next step in this workflow would be to implement the new component. You can use rapid prototyping and embedded code generation products to generate code and use the design with a real, physical system.

See Also

Related Examples

- “System Definition and Layout” on page 1-8
- “Model and Validate a System” on page 1-16
- “Design a System in Simulink” on page 1-29
- “Organize Large Modeling Projects”

External Websites

- Simulink Overview
- Model-Based Design with MATLAB and Simulink

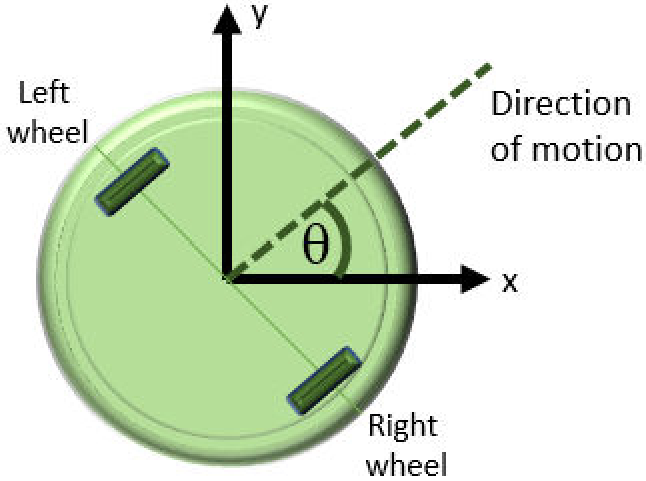
System Definition and Layout

In this section...

The top-level system layout of a Simulinkmodel is a common context that many engineering teams can use, and is the basis for many tasks in the Model-Based Design paradigm: Analysis, design, test, and implementation. You define a system at the top level by identifying the structure and individual components. You then organize your model in a hierarchical manner that corresponds to the components. Then you define interfaces for each component, and the connections between components.

The featured model is a flat robot that can move or rotate with the help of two wheels, similar to a home vacuuming robot. This tutorial assumes that the robot moves in one of two ways:

- Linear — Both wheels turn in the same direction with the same speed, and the robot moves linearly.
- Rotational — The wheels turn in opposite directions with the same speed, and the robot rotates in place.



Each type of motion starts from a resting state, that is, both rotational and linear speeds are zero. With these assumptions, linear and rotational motion components can be modeled separately for this introductory tutorial.

Determine Modeling Objectives

Before designing a model, consider your goals and requirements. The goals dictate both the structure, and the level of detail for the model. For example, if the goal is simply to figure out how fast the robot can go, modeling just for linear motion is sufficient. If the goal is to design a set of inputs for the device to follow a given path, then the rotational component is involved. If obstacle avoidance is a goal, then the system needs a sensor. This tutorial builds a model for the goal of designing sensor parameters so that the robot stops in time when it detects an obstacle on its path. To achieve this goal, the model must enable you to:

- Determine how quickly the robot stops when the motors stop
- Provide a series of commands for linear and rotational motion so that it can move over a two-dimensional space

The first modeling objective enables you to analyze the motion so you can design the sensor. The second objective enables you to test your design.

Identify System Components and Interfaces

Once you understand your modeling requirements, you can begin to identify the components of the system. Identifying individual components and their relationships within a top-level structure help build a potentially complex model systematically. You perform these steps outside Simulink before you begin building your model.

This task involves answering these questions:

- What are the structural and functional components of the system? When a layout reflects the physical and functional structure, it helps to understand, build, communicate, and test the system. This becomes more important when parts of the system are to be implemented in the process.
- What are the inputs and outputs for each component? Draw a picture showing the connections between components. This picture leads to signal flow within the model, and, in addition to the source and sink of each signal, it helps determine if all necessary components exist.

- What level of detail is necessary? Include major parameters in your diagram. Creating a picture of the system can help you identify and model the parts that are essential to the behaviors you want to observe. Each component and parameter that contributes to the goal must have a representation in the model, but there is a tradeoff between complexity and readability. Modeling can be an iterative process: You can start with a high-level model with few details, and gradually increase complexity where required.

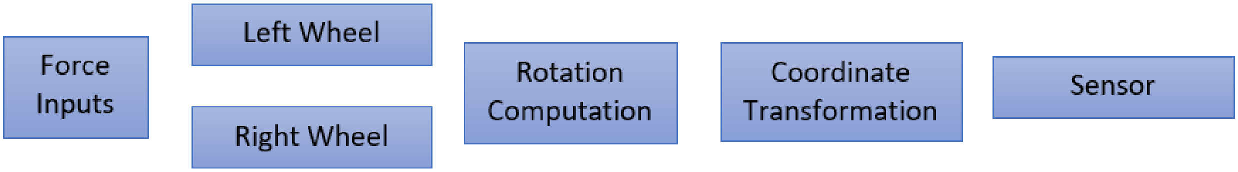
In addition, it is often beneficial to consider the following:

- What parts of the system need testing?
- What is the test data and success criteria?
- Which outputs are necessary for analysis and design tasks?

Identify Robot Motion Components

The system in this tutorial defines a robot that moves with two electric wheels in two dimensions. It includes:

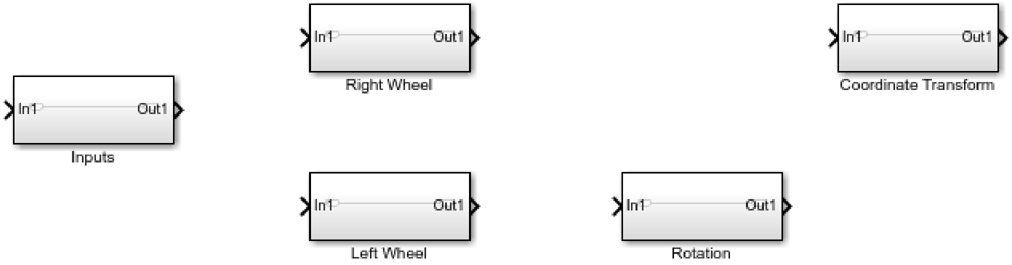
- Linear motion characteristics
- Rotational motion characteristics
- Transformations to determine the location of the system in two dimensions
- A sensor to measure the distance of the robot from an obstacle



The model for this system includes two identical wheels, input forces applied to the wheels, rotational dynamics, coordinate transformation, and a sensor. The model uses a Subsystem to represent each component.

1. Open a new Simulink model: “Open New Model” on page 3-3.
2. From the **Display** menu, clear the **Hide Automatic Names** check box.
3. Open the Library Browser: “Open Simulink Library Browser” on page 3-5
4. Add Subsystem blocks. Drag five Subsystem blocks from the Ports & Subsystems library to the new model.

Arrange and rename the Subsystem blocks as shown. Double-click a block name and type the new name.



Define Interfaces Between Components

Identify input and output connections (for example, signal lines) between subsystems. Input and output values change dynamically during a simulation. Lines connecting blocks represent data transfer. The table below shows the inputs and outputs for each component.

Block	Input	Output	Notes
Inputs	None	Force to right wheel Force to left wheel	
Right wheel	Force to right wheel	Right wheel velocity	Directional, negative means reverse direction
Left wheel	Force to left wheel	Left wheel velocity	Directional, negative means reverse direction
Rotation	Velocity difference between right and left wheels	Rotational angle	Measured counterclockwise
Coordinate transformation	Normal speed Rotational angle	Velocity in X Velocity in Y	

Block	Input	Output	Notes
Sensor	X coordinate Y coordinate	None	No block necessary for modeling. Sensor dynamics is part of the design task.

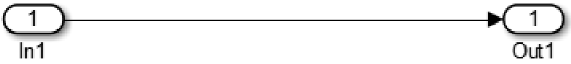
From the table, you can see that some block inputs do not exactly match block outputs. Therefore, in addition to the dynamics of the individual components, the model must compute the following:

- Input to the rotation computation — Subtract the velocities of the two wheels and divide by two.
- Input to the coordinate transformation — Average the velocities of the two wheels.
- Input to the sensor — Integrate the outputs of the coordinate transformation.

The wheel velocities are always equal in magnitude and the computations are accurate within that assumption.

Add the necessary components and finalize connections:

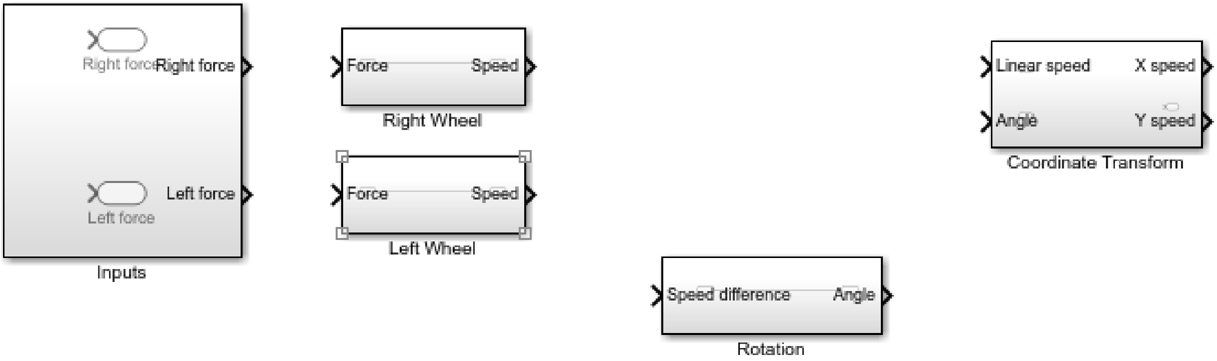
1 Add necessary input and output ports to each subsystem. Double-click a Subsystem block.



Each new Subsystem block contains one Inport (In1) and one Outport (Out1) block. These blocks define the signal interface with the next higher level in a model hierarchy.

Each Inport block creates an input port on the Subsystem block, and each Outport block creates an output port. The model reflects the names of these blocks as the input/output port names. Add more blocks for additional input and output signals. On the Simulink Editor toolbar, click the **Up to Parent** button ⇧ to return to the top level.

For each block, add and rename Inport and Outport blocks:

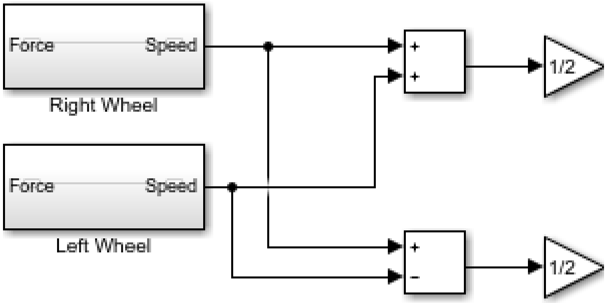


When copying an Inport block to create a new one, you must use the **Paste** option.

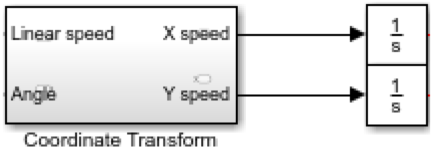
2 Compute required inputs from left wheel and right wheel velocities shown.

   a Add an Add block from the Math Operations library and connect the outputs of the two-wheel components. Click the output port of the source block and then click the input port of the destination block. Add a Gain block and set the parameter to `1/2`. Compute the Linear speed input to the Coordinate Transform subsystem, connect the output of the Add block to this Gain block.

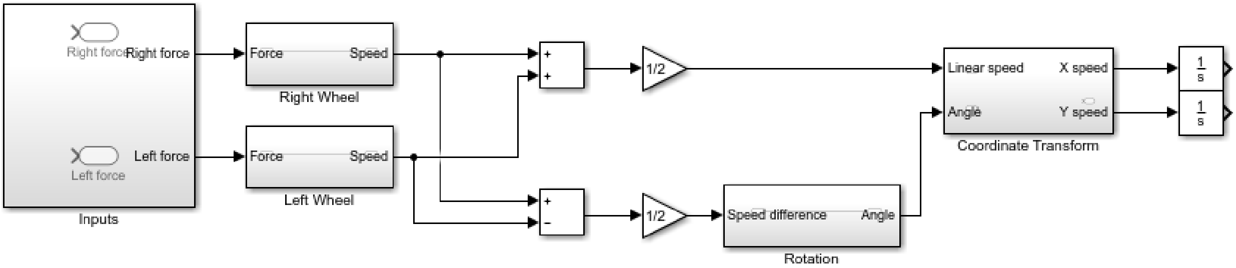
   b Add a Subtract block from the Math Operations library and connect the outputs of the two-wheel components. Add a Gain block and set the parameter to `1/2`. Compute the Speed difference input to the Rotation subsystem, connect the output of the Subtract block to this Gain block.



3 Compute X and Y coordinates from the X and Y velocities. Add two Integrator blocks from the Continuous library and connect the outputs of the Coordinate Transform block. Leave initial conditions to the Integrator blocks as `0`.



4 Complete the connections for the system as shown.



Parameters and Data

Determine the parameters that are part of the model and their values. Use modeling goals to determine whether these values are always fixed or change from simulation to simulation. Parameters that contribute to the modeling goal require explicit representation in the model. This table helps determine the level of detail when modeling each component.

Parameter	Block	Symbol	Value/Unit	Notes
Mass	Left/right wheel	m	2.5 kg	Variable
Rolling resistance	Left/right wheel	k_drag	30 Ns^2/m	Variable
Robot radius	Rotation computation	r	0.15 m	Variable
Initial angle	Rotation computation	None.	0	Fixed
Initial velocities	Left/right wheel	None.	(0,0)	Fixed
Initial coordinates	Integrators	None	(0,0)	Fixed

Simulink uses MATLAB workspace to evaluate parameters. Set these parameters in the MATLAB command window:

```
m = 2.5;
k_drag = 30;
r = 0.15;
```

See Also

Related Examples

- “Model and Validate a System” on page 1-16
- “Design a System in Simulink” on page 1-29

Model and Validate a System

You model each component within the system structure to represent the physical or functional behavior of that component. You verify the basic component behavior by simulating them using test data.

A big-picture view of the whole system layout is useful when modeling individual components. Start by loading the layout model:

```
open_system(fullfile(matlabroot,...
'help', 'toolbox', 'simulink', 'examples', 'system_layout'))
```

Model the Components

A Simulink model of a component is based on several starting points:

- An explicit mathematical relationship between the output and the input of a physical component — You can compute the outputs of the component from the inputs, directly or indirectly, through algebraic computations and integration of differential equations. For example, computation of the water level in a tank given the inflow rate is an explicit relationship. Each Simulink block executes based on the definition of the computations from its inputs to its outputs.
- An implicit mathematical relationship between model variables of a physical component — Because variables are interdependent, assigning an input and an output to the component is not straightforward. For example, the voltage at the + end of a motor connected in a circuit and the voltage at the – end have an implicit relationship. To model such a relationship in Simulink, you can either use physical modeling tools such as Simscape™ or model these variables as part of a larger component that allows input/output definition. Sometimes, closer inspection of modeling goals and component definitions helps to define input/output relationships.
- Data obtained from an actual system — You have measured input/output data from the actual component, but a fully defined mathematical relationship does not exist. Many devices have unmodeled components that fit this description. An example would be the heat a TV set dissipates. You can use System Identification Toolbox™ to define the input/output relationship for such a system.
- An explicit functional definition — You define the outputs of a functional component from the inputs through algebraic and logical computations. The switching logic of a thermostat is an example. You can model most functional relationships as Simulink blocks and subsystems.

This tutorial models physical and functional components with explicit input/output relationships:

1. Use the system equations to create a Simulink model.
2. Add Simulink blocks in the Simulink Editor. Blocks represent coefficients and variables from the equations. Connect blocks.
3. Build the model for each component separately. The most effective way to build a model of a system is to consider components independently.
4. Start by building simple models using approximations of the system. Identify assumptions that can affect the accuracy of your model. Iteratively add detail until the level of complexity satisfies the modeling and accuracy requirements.

Model the Physical Components

Describe the relationships between components, for example, data, energy, and force transfer. Use the system equations to build a graphical model of the system in Simulink.

Some questions to ask before you begin to model a component:

- What are the constants for each component and the values that do not change unless you change them?
- What are the variables for each component and the values that change over time?
- How many state variables does a component have?

Derive the equations for each component using scientific principles. Many system equations fall into three categories:

- For continuous systems, differential equations describe the rate of change for variables with the equations defined for all values of time. For example, a second-order differential equation provides the velocity of a car:
- For discrete systems, difference equations describe the rate of change for variables, but the equations are defined only at specific times. For example, the following difference equation gives the control signal from a discrete propositional-derivative controller:
- Equations without derivatives are algebraic equations. For example, an algebraic equation gives the total current in a parallel circuit with two components:

Wheels and Linear Motion

There are two forces that act on a wheel:

- Force applied by the motor — This force F acts in the direction of velocity change, and is an input to the wheel subsystem.
- Drag force — This force F_{drag} acts against the direction of velocity change, and is a function of the velocity itself:

  $$F_{drag} = k_{drag}V|V|$$

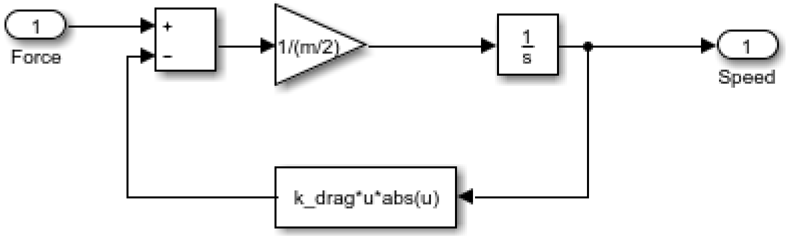
The acceleration is proportional to the sum of these forces:

$$\begin{aligned}(m/2)\dot{V} &= F - F_{drag}\\ (m/2)\dot{V} &= F - k_{drag}V|V|\\ \dot{V} &= \frac{F - k_{drag}V|V|}{(m/2)}\end{aligned}$$

Where k_{drag} is the drag coefficient and m is the mass of the robot. Each wheel carries half of this mass.

Build the wheel model:

1. In the layout model, double-click the Right Wheel subsystem to display the empty subsystem. Delete the connection between the Inport and the Outport blocks.
2. Model velocity and acceleration. Add an Integrator block. Leave the initial condition as `0`. The output of this block is the velocity, *V*, and the input is the acceleration, *Vdot*.
3. Model the drag force. Add an Fcn block from the User-Defined Functions library. Set the expression to `k_drag*u*abs(u)`. You can resize the block to see the expression on its icon. The Fcn block provides a quick way to type simple mathematical expressions of one input variable, *u*.
4. Subtract the drag force from the motor force with Subtract block, and complete the force-acceleration equation with a Gain block with parameter `1/(2*m)`.
5. To reverse the direction of the Fcn block, right-click the block and select **Rotate & Flip > Flip Block**. Make the connections between blocks as shown.



6

View the top level of the model: Click the **Up to Parent** button ⇧. Make a copy of the subsystem you modeled as the dynamics for both wheels are the same.

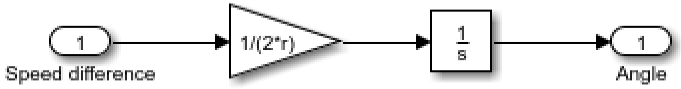
Rotational Motion

When the two wheels turn in opposite directions, that is, they have directionally opposite velocities, they move in a circle with radius *r*, causing rotational motion. When they turn in the same direction, there is no rotation. Therefore, with the assumption that the wheel velocities are always the same in magnitude, it is practical to model rotational motion as dependent on the difference of the two velocities, V_R and V_L:

$$\dot{\theta} = \frac{V_R - V_L}{2r}$$

Build the Rotation Dynamics model:

1. In the layout model, double-click the Rotation subsystem to display the empty subsystem. Delete the connection between the Inport and the Outport.
2. Model angular speed and angle: Add an Integrator block. Leave the initial condition as `0`. The output of this block is the angle, theta, and the input is the angular speed, *theta_dot*.
3. Compute angular speed from tangential speed. Add a Gain with parameter `1/(2*r)`.
4. Make the connections between blocks as follows.



5 View the top level: Click the **Up to Parent** button ⇧.

Model the Functional Components

Describe the function from the input of a function to its output. This description can include algebraic equations and logical constructs, which you can use to build a graphical model of the system in Simulink.

Coordinate Transformation

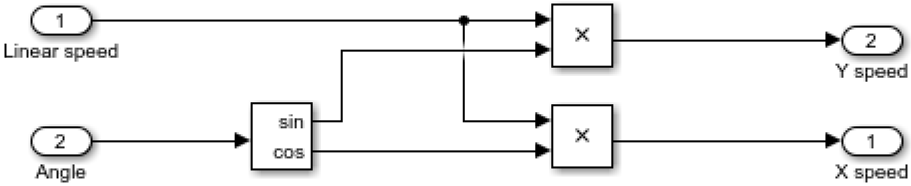
The velocity of the robot in the X and Y coordinates, Vx and Vy, are related to the linear speed, Vn, and the angle as follows:

$$V_X = -V_N\cos(\theta)$$

$$V_Y = V_N\sin(\theta)$$

Build coordinate transformation model:

1. In the layout model, double-click the Coordinate Transform subsystem to display the empty subsystem.
2. Model trigonometric functions. Add a SinCos block from the Math Operations library.
3. Model multiplications. Add two Product blocks from the Math Operations library.
4. Make connections between the blocks as shown.



5. View the top level: Click the **Up to Parent** button ⇧.

Set Model Parameters

A source for model parameter values can be:

- Written specifications such as standard property tables or manufacturer data sheets

- Direct measurements on an existing system
- Estimations using system input/output

The model uses these parameters:

Parameter	Symbol	Value/Unit
Mass	m	2.5 kg
Rolling resistance	k_drag	30 Ns^2/m
Robot radius	r	0.15 m

Simulink uses MATLAB workspace to evaluate parameters. Set these parameters in the MATLAB command window:

```
m = 2.5;
k_drag = 30;
r = 0.15;
```

Validate Components Using Simulation

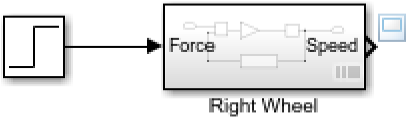
Validate components by supplying an input and observing the output. Even such a simple validation can point out immediate ways to improve the model. This example validates the following behavior:

- When a force is applied continuously to a wheel, the velocity increases until it reaches a steady-state velocity.
- When the wheels are turning in opposite directions, the angle increases steadily.

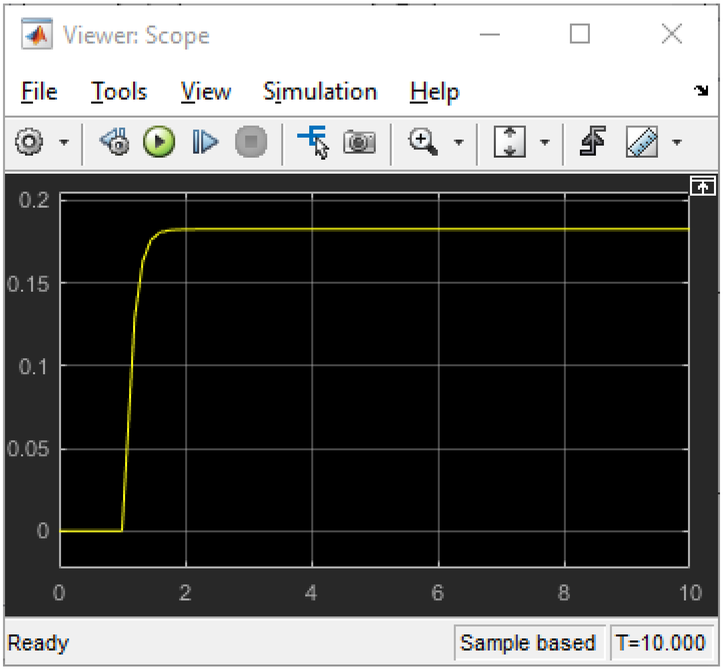
Validate Wheel Component

Create and run a test model for the wheel component:

1. Create a model. Click and copy the Right Wheel block into the new model.
2. Create a test input in the new model. Add a Step block from the Sources library. Connect it to the input of the Right Wheel block.
3. Add a viewer to the output. Right-click the output port of the Right Wheel block and select **Create & Connect Viewer > Simulink > Scope**.



4 Run the simulation. Click ▶.

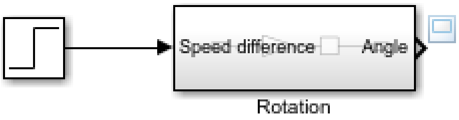


The simulation result exhibits the general expected behavior. There is no motion until force is applied at step time. When force is applied, the speed starts increasing and then settles at a constant when the applied force and the drag force reach an equilibrium. Besides validation, this simulation also gives information on the maximum speed of the wheel with the given force.

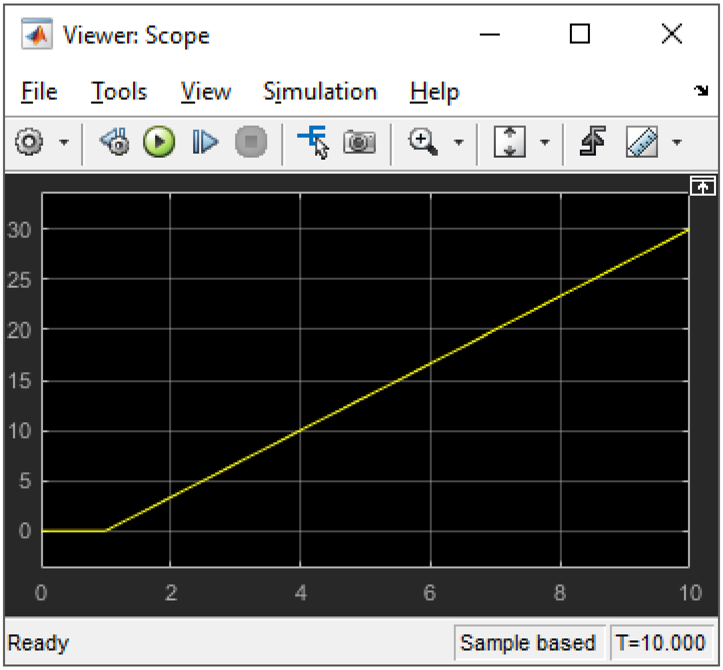
Validate Rotation Component

Create and run a test model for the rotation model:

1. Create a model. Click and copy the Rotation block into the new model.
2. Create a test input in the new model. Add a Step block from the Sources library. Connect it to the input of the Rotation block. This input represents the difference of the wheel velocities when the wheels are rotating in opposite directions.
3. Add a viewer to the output. Right-click the output port of the Rotation block and select **Create & Connect Viewer > Simulink > Scope**.



4. Run the simulation. Click .



This simulation shows that the angle increases steadily when the wheels are turning with the same speed in opposite directions. You can make some model improvements to make it easier to interpret the angle output, for example:

- You can convert the output in radians to degrees. Add a Gain block with a gain of `180/pi`.
- You can display the degrees output in cycles of 360 degrees. Add a Math Function block with function `mod`.

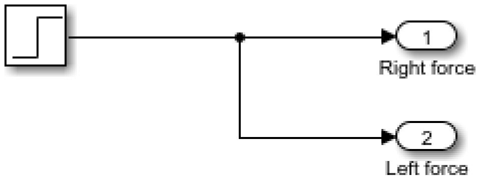
MATLAB trigonometric functions take inputs in radians.

Validate the Model

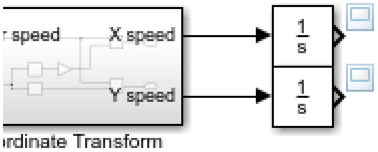
After you validate components, you can perform a similar validation on the complete model. This example validates the following behavior:

- When the same force is applied to both wheels in the same direction, the vehicle moves in a line.
- When the same force is applied to both wheels in opposite directions, the vehicle turns around itself.

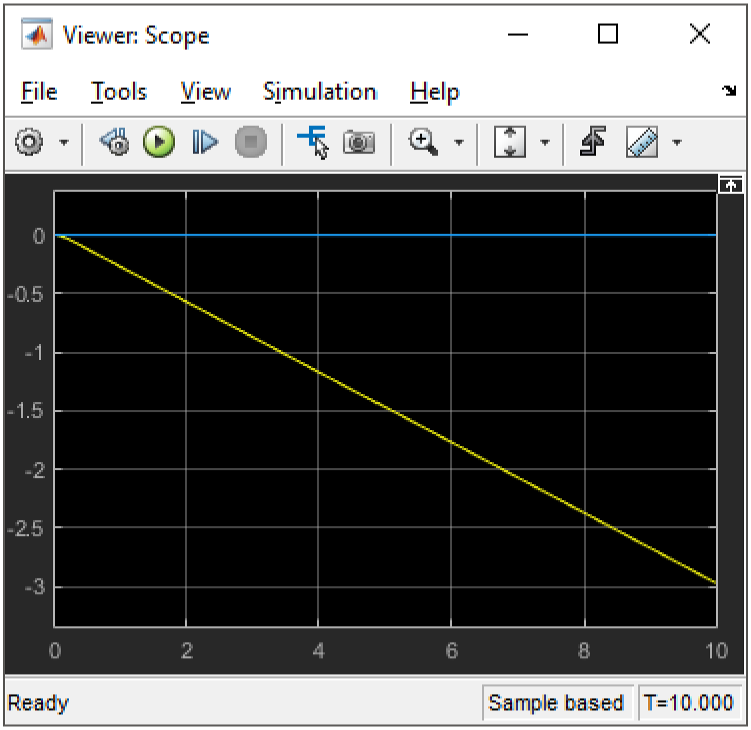
1. In the layout model, double-click the Inputs subsystem to display the empty subsystem.
2. Create a test input by adding a Step block. Connect it to both Outport blocks.



3. At the top level of the model, add both output signals to the same viewer:

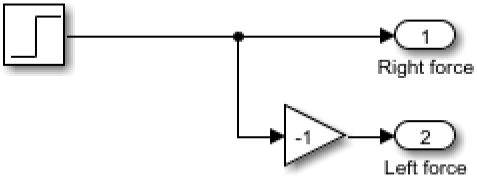


4. Run the model.



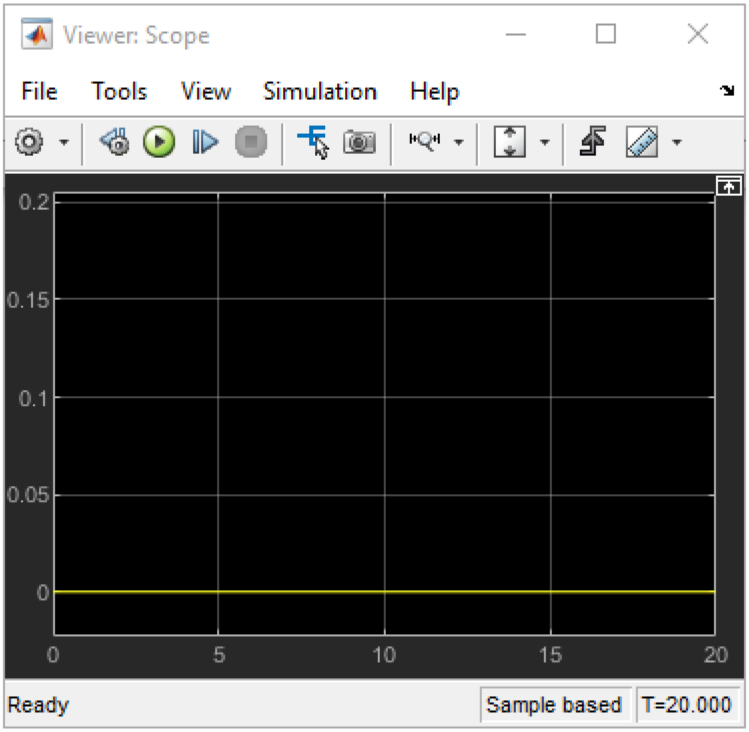
In this figure, the yellow line is the X direction and the blue line is the Y direction. Since the angle is zero and is not changing, the vehicle moves only in the X direction, as expected.

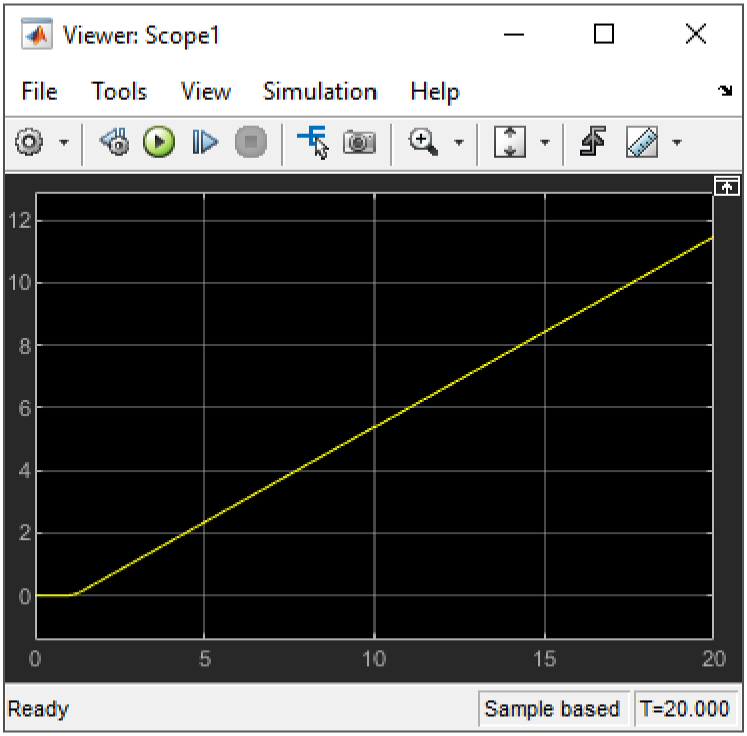
5 Double-click the Inputs subsystem and add a Gain with parameter `-1` between the source and the second output. This reverses the direction for the left wheel.



6 Add a scope to the angle output.

7 Run the model.


Viewer: Scope
File
Tools
View
Simulation
Help
0.2
0.15
0.1
0.05
0
0
5
10
15
20
Ready
Sample based
T=20.000



The first view shows that there is no motion in the X-Y plane. The second view shows that there is steady rotation.

You can use this final model to answer many questions about the model by changing the input. Some examples are:

- What happens when the initial angle is not zero?
- How long does it take for the motion to stop when the force drops to zero?
- What happens when the robot is heavier?
- What happens when the robot moves on a smoother surface, that is, the drag coefficient is lower?

See Also

Related Examples

- “System Definition and Layout” on page 1-8
- “Design a System in Simulink” on page 1-29

Design a System in Simulink

In this section...

Model-Based Design paradigm is centered around models of physical components and systems as a basis for design, testing, and implementation activities. This tutorial adds a designed component to an existing system model.

The model is a flat robot that can move or rotate with the help of two wheels, similar to a home vacuuming robot. Open the model by entering the code at the MATLAB command line.

```
open_system(fullfile(matlabroot,...
'help', 'toolbox', 'simulink', 'examples', 'system_model'))
```

This tutorial analyzes this system and adds functionality to it.

Identify Designed Components and Design Goals

Proper specification of the objective is a critical first step to the design task. Even with a simple systems, there could be multiple, and even competing design goals. Consider these for the example model:

- Design a controller that varies the force input so that the wheels turn at a desired speed.
- Design inputs that make the device move in a predetermined path.
- Design a sensor and a controller so that the device follows a line.
- Design a planning algorithm so that the device reaches a certain point using the shortest path possible while avoiding obstacles.
- Design a sensor and an algorithm so that the device moves over a certain area while avoiding obstacles.

This tutorial designs an alert system. You determine the parameters for a sensor that measures the distance from an obstacle. A perfect sensor measures the distance from an obstacle accurately, an alert system samples those measurements at fixed intervals so that the output is always within 0.05 m of the measurement, and generates an alert in time for the robot to come to a stop.

Analyze System Behavior Using Simulation

The design of the new component requires analyzing linear motion to determine:

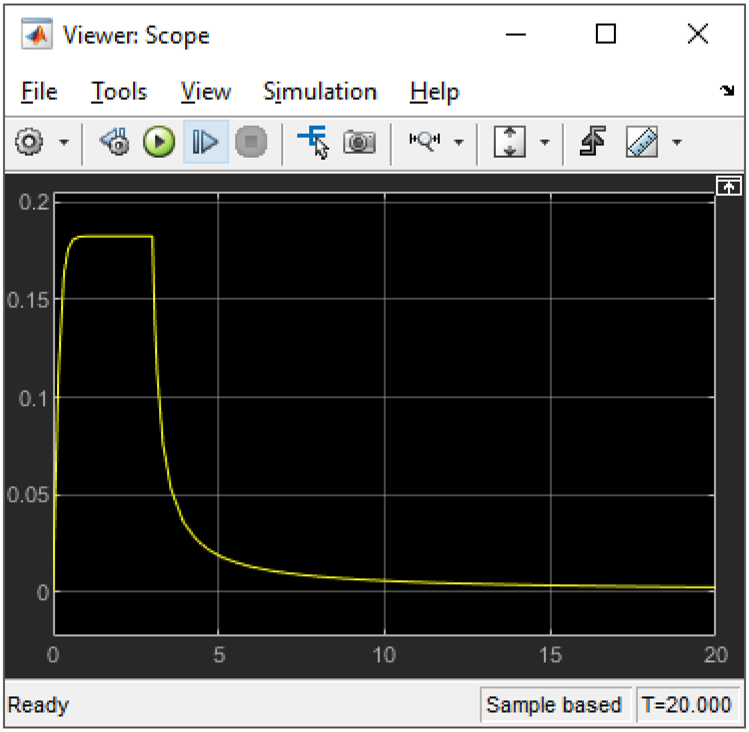
- How far the robot can travel at the top speed if power to the wheels is cut
- The robot's top speed

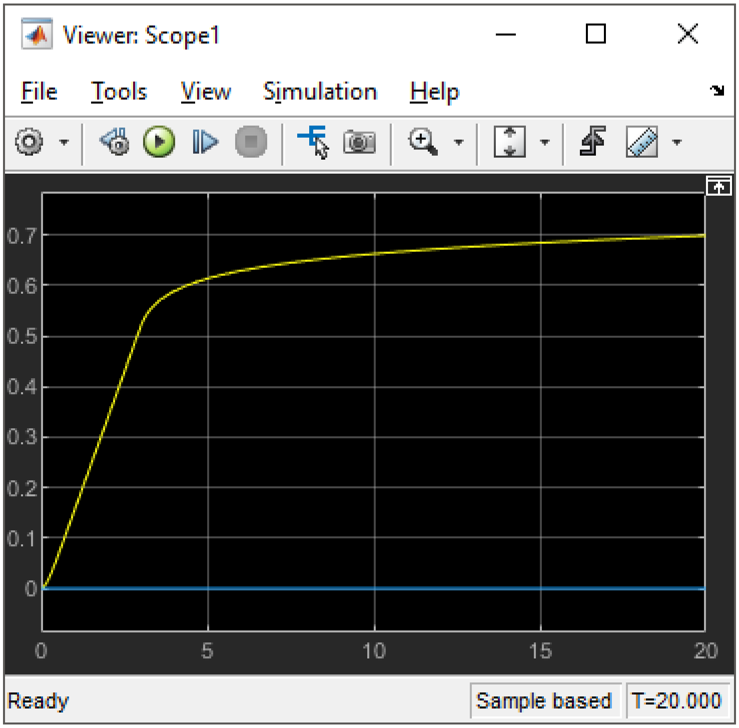
Run the model with a force input that starts motion, waits until the robot reaches a steady velocity, and then sets the force to zero:

1. In the model, double-click the Inputs subsystem.
2. Delete the existing input and add a Pulse Generator block with the default **Amplitude** parameter.
3. Set parameters for the Pulse Generator block:

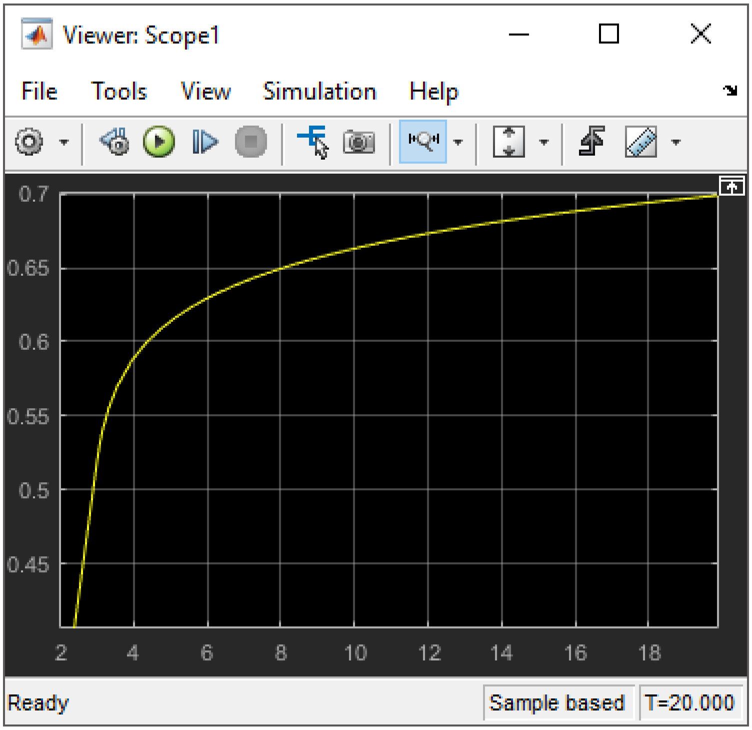
   - Period: `20`
   - Pulse Width: `15`

   These parameters are designed to ensure that the top speed is reached. You can change parameters to see their effect.
4. Run the model for 20 sec.


Viewer: Scope
File
Tools
View
Simulation
Help
0.2
0.15
0.1
0.05
0
0
5
10
15
20
Ready
Sample based
T=20.000



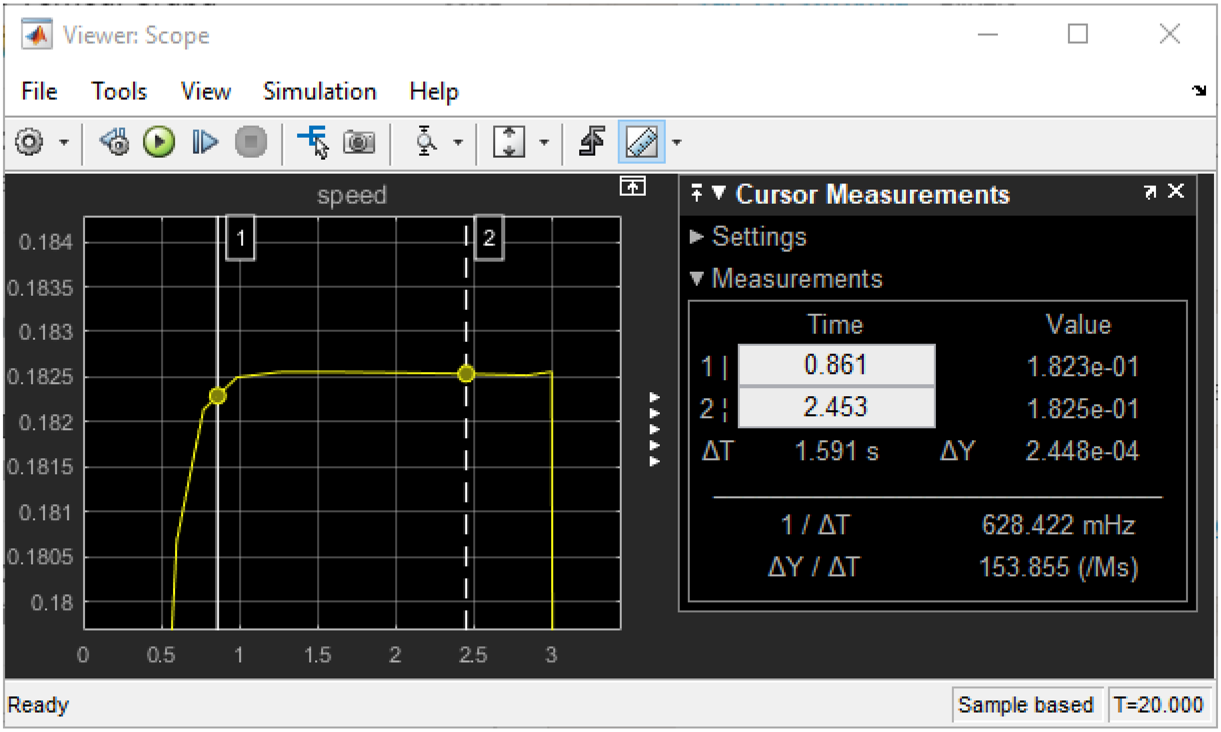
The first scope shows that the speed rapidly starts decreasing when the power is cut at time 3, and then asymptotically approaches zero but does not quite reach it. This is a limitation of modeling — the dynamics at low speeds without external force may require a more complex representation. For the objective here, however, it is possible to make approximations. Zoom into the position signal



At time 3, the position of the robot is at about 0.55 m, and when the simulation ends, it is less than 0.71 m. It is safe to say that the robot travels less than 0.16 m after the power is cut.

To find the top speed,

1 Zoom on the stable region of the velocity output in time, from 1 s to 3 s.
2 Leave zoom mode by clicking the zoom button again. Click Cursor Measurements button.
3 Set the second cursor to the region where the line is horizontal.



	Time		Value
1	0.861		1.823e-01
2	2.453		1.825e-01
ΔT	1.591 s	ΔY	2.448e-04
1 / ΔT			628.422 mHz
ΔY / ΔT			153.855 (/Ms)

The **Value** column in Cursor Measurements indicate that the top speed of the robot is 0.183 m/s. Divide 0.05 by this speed to obtain the time it takes the robot to travel 0.05 m — 0.27 s.

Design Components and Verify Design

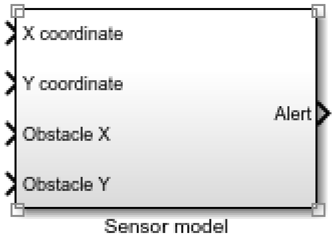
Sensor design consists of these components:

- Measurement of the distance between the robot and the obstacle — This example assumes that the measurement is perfect.
- The interval at which the sensor system measures the distance: To keep the measurement error below 0.05 m, this interval should be less than 0.27 sec. Use 0.25 sec.
- The distance at which the sensor produces an alert — Analysis shows that slow down must start at 0.16 m, but the actual alert distance must also take the measurement error, 0.05, into account.

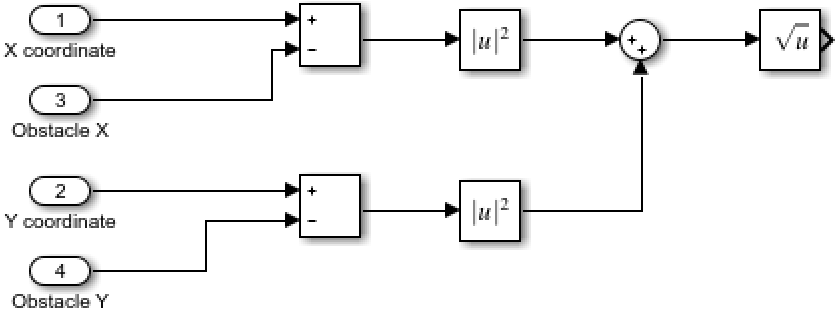
Add Designed Component

Build the sensor:

1. Create a subsystem with the ports as shown.

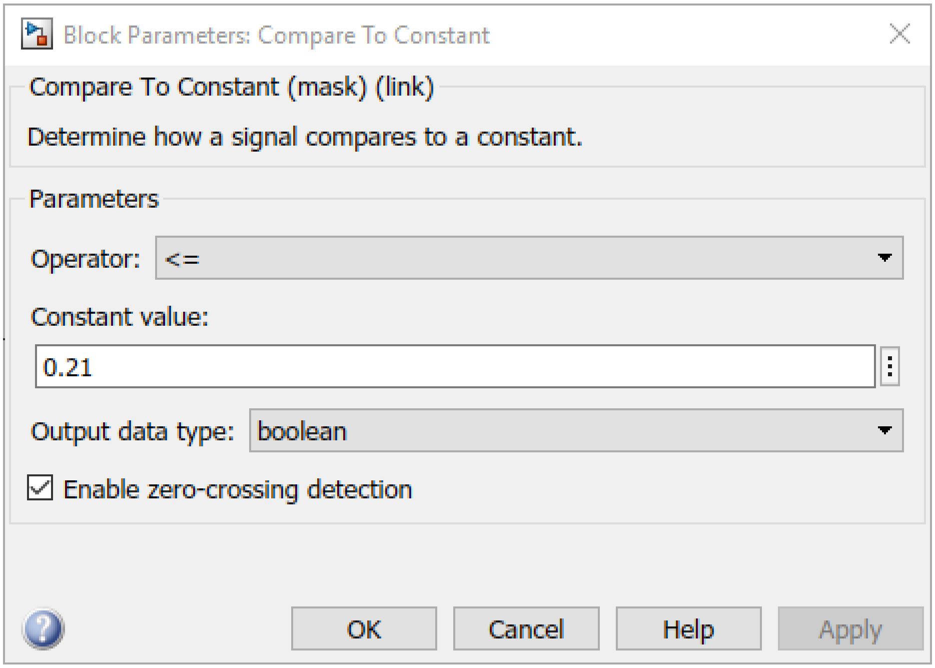


2. Construct the distance measurement. In the sensor model block, use Subtract, Math Function with `magnitude^2` function, Sum, and Sqrt blocks as shown. Note the reordering of the input ports.



3. Model sampling. Add a Zero-Order Hold block from the Discrete library to the subsystem and set its **Sample Time** parameter to `0.25`.
4. Model the alert logic. Use the Compare to Constant from Math Operations and set its parameters:

   - **Operator**: <=
   - **Constant Value**: `0.21`



Block Parameters: Compare To Constant

Compare To Constant (mask) (link)

Determine how a signal compares to a constant.

Parameters

Operator: <=

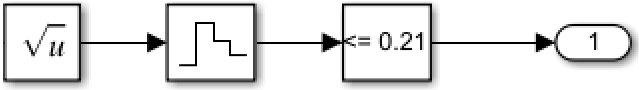
Constant value:

0.21

Output data type: boolean

☑ Enable zero-crossing detection

This logical block sets its output to `1` when its input is less than `0.21`.

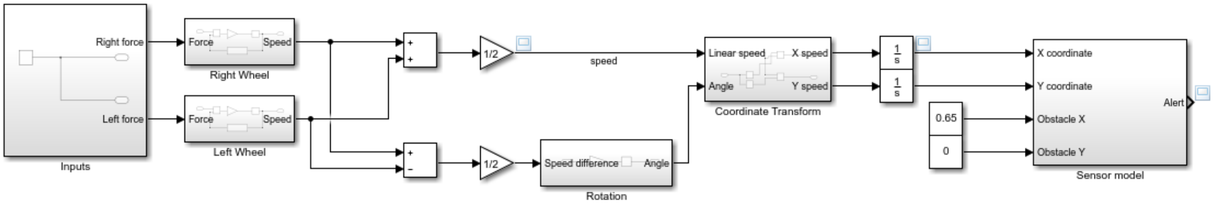


Verify Design

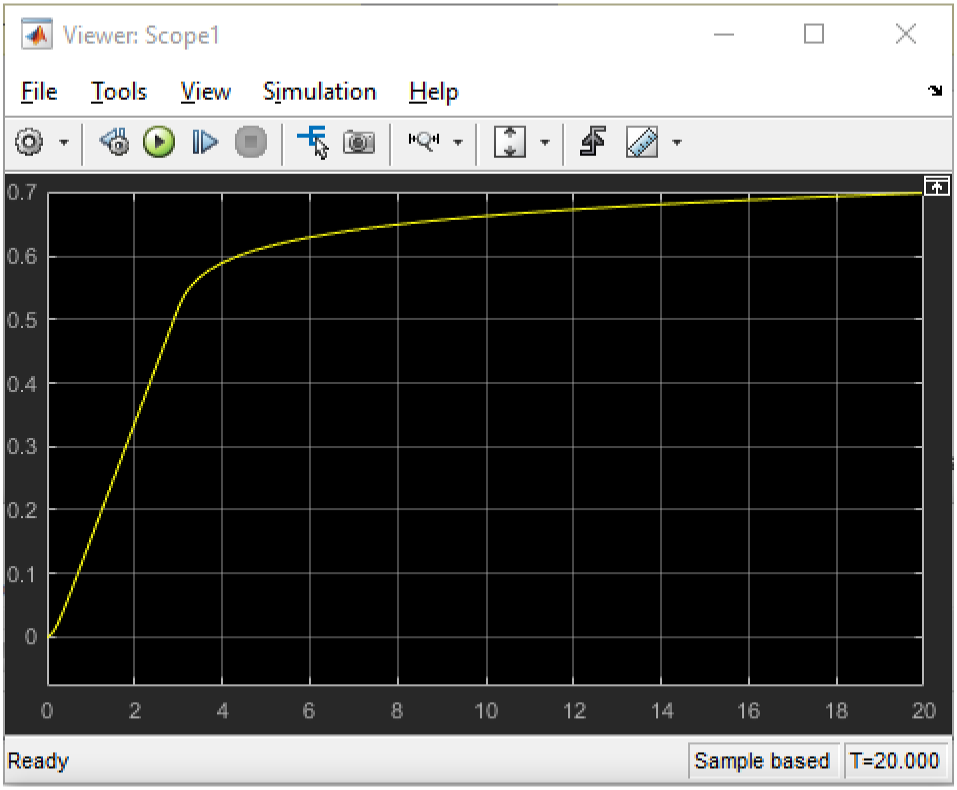
Test the design with an obstacle location of X=0.65, Y=0, using Constant blocks as input. This test verifies functionality in the X direction, you can create similar tests for different paths. This model only generates an alert. It does not control the robot.

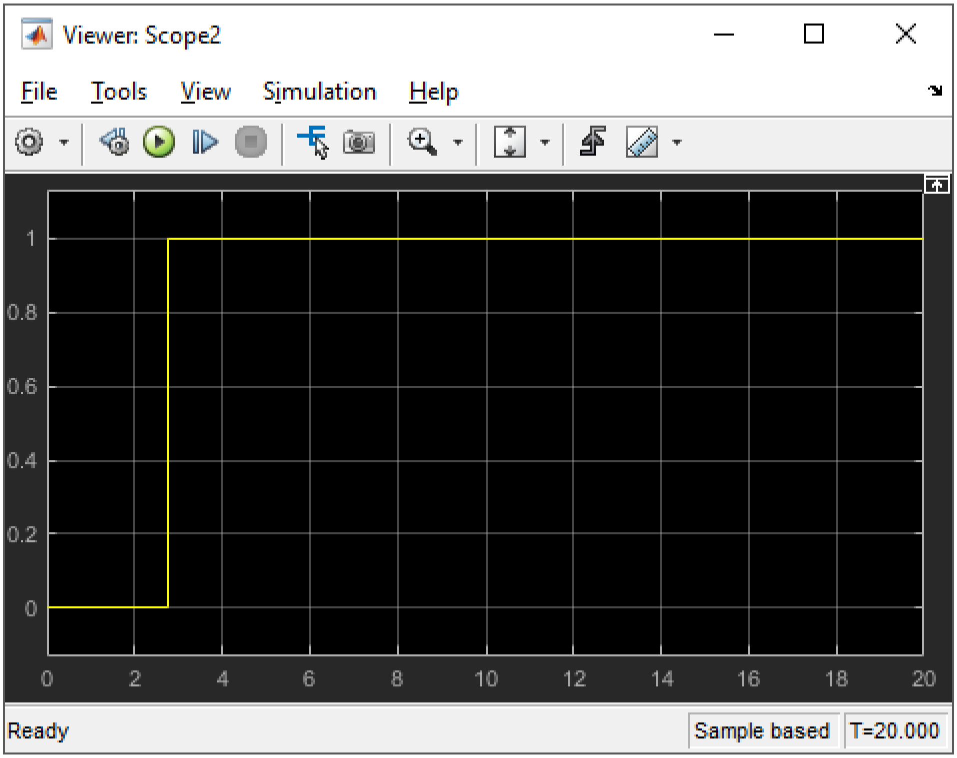
1 Set the obstacle location: Add two Constant blocks from the Sources library set the constant values to `0.65` and `0`. Connect the position outputs of the robot to the inputs of the sensor.

2 Add a scope to the Alert output.



3 Run the model.





Observe that the alert status becomes 1 once the position is within 0.21 m of the obstacle location and the design requirement for this component is satisfied.

For real-world systems with complex components and formal requirements, the Simulink product family includes additional tools refine and automate the design process. Simulink Requirements™ provide tools to formally define requirements and link them to model components. Simulink Control Design™ can facilitate the design if you want to build a controller for this robot. Simulink Verification and Validation™ products establish a formal framework for testing components and systems.

See Also

Related Examples

- “Model-Based Design with Simulink” on page 1-3
- “System Definition and Layout” on page 1-8
- “Model and Validate a System” on page 1-16

Documentation and Resources

In this section...

Simulink Online Help

Simulink software provides comprehensive online help describing features, blocks, and functions with detailed procedures for common tasks.

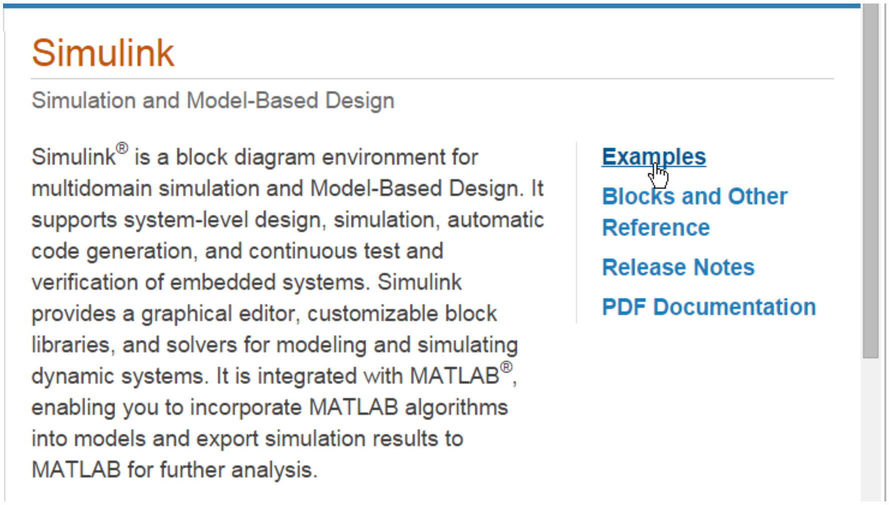
Access online help from **Help** menus and context-sensitive block labels.

- From the Simulink Library Browser toolbar, select the **Help** button ?.
- From the Simulink Editor menu, select **Help** > **Simulink** > **Simulink Help**.
- Right-click a Simulink block, and then select **Help**.
- From the model Configuration Parameters dialog box or a block parameters dialog box, right-click a parameter label, then select **What's This?**

Simulink Examples

Simulink provides example models that illustrate key modeling concepts and Simulink features. To view a list of examples:

- From the Simulink Editor menu, select **Help** > **Simulink** > **Examples**.
- From the Help browser, open the Simulink product page, and then click **Examples** at the top right.

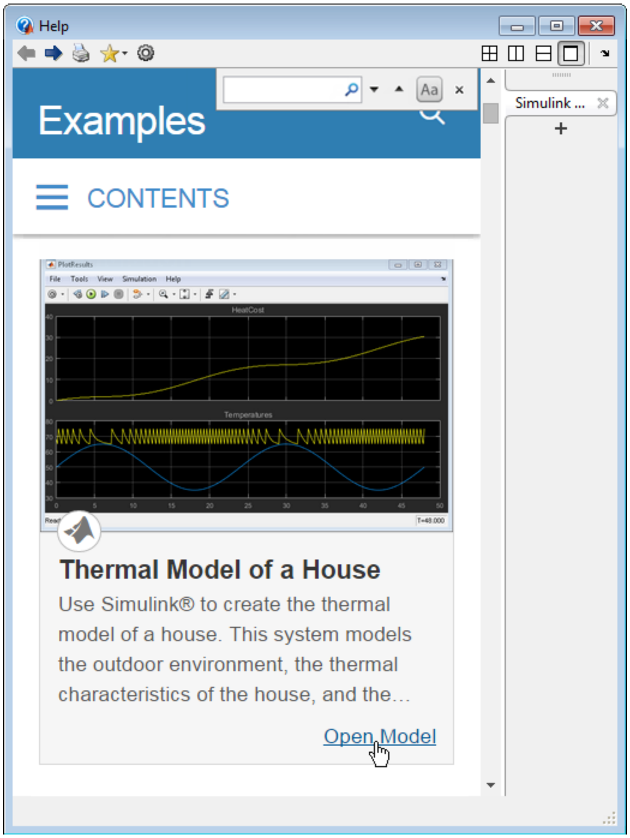


Simulink

Simulation and Model-Based Design

Simulink® is a block diagram environment for multidomain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

To open the Simulink model for an example, click the **Open Model** button.



Thermal Model of a House

Use Simulink® to create the thermal model of a house. This system models the outdoor environment, the thermal characteristics of the house, and the…

Website Resources

You can access additional Simulink resources on the MathWorks website, including a description of capabilities, technical articles, tutorials, and hardware support.

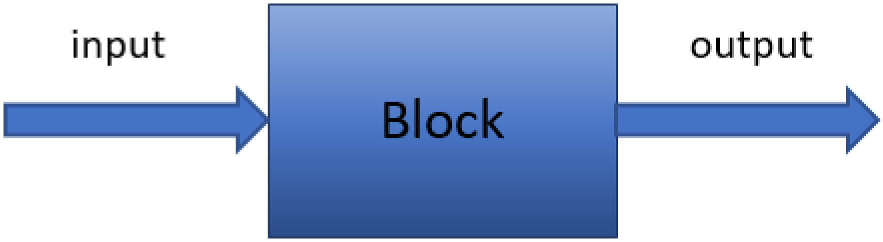
```
https://www.mathworks.com/products/simulink
```

2

Modeling in Simulink

Simulink Block Diagrams

Simulink is a graphical modeling and simulation environment for dynamic systems. You can create block diagrams, where blocks represent parts of a system:

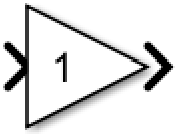


A block can represent a physical component, a small system, or a function; an input/output relationship fully characterizes the block. Consider these examples:

- A faucet fills a bucket: Water goes into the bucket at a certain flow rate, and the bucket gets heavier. Here, a block represents the bucket, with flow rate as its input and its weight as the output.
- You use a megaphone to make your voice heard: Sound produced at one end of the megaphone is amplified at the other end. The megaphone is the block, the input is the sound wave at its source, and the output is the sound wave as you hear it.
- You push a cart and it moves: Here the cart can be the block, the force you apply is the input and cart position is the output.

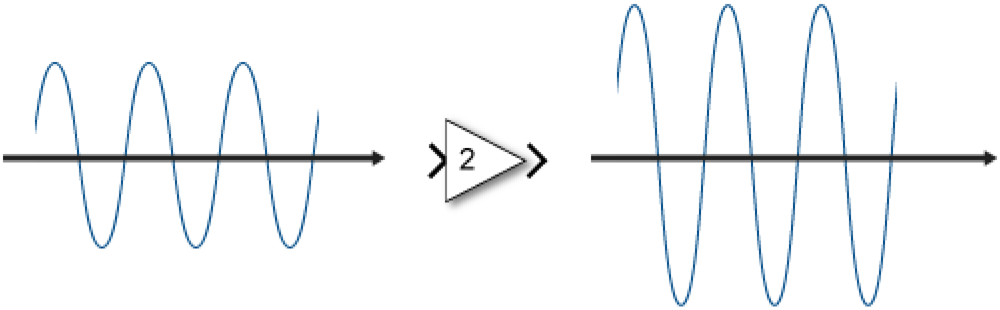
The definition of a block is only complete with its inputs and outputs and this task relates to the goal of the model. For example, the cart velocity may be a natural choice as an output if the modeling goal does not involve its location.

Simulink provides block libraries that are collections of blocks grouped by functionality. For example, to model a megaphone that simply multiplies its input by a constant, you would use a Gain block from the Math Operations library.



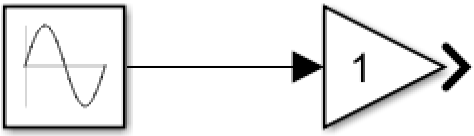
A sound wave goes into the megaphone, as its input, and a louder version of the same wave comes out as its output.

The ">" signs denote the inputs and outputs of a block, and can be connected to other blocks.

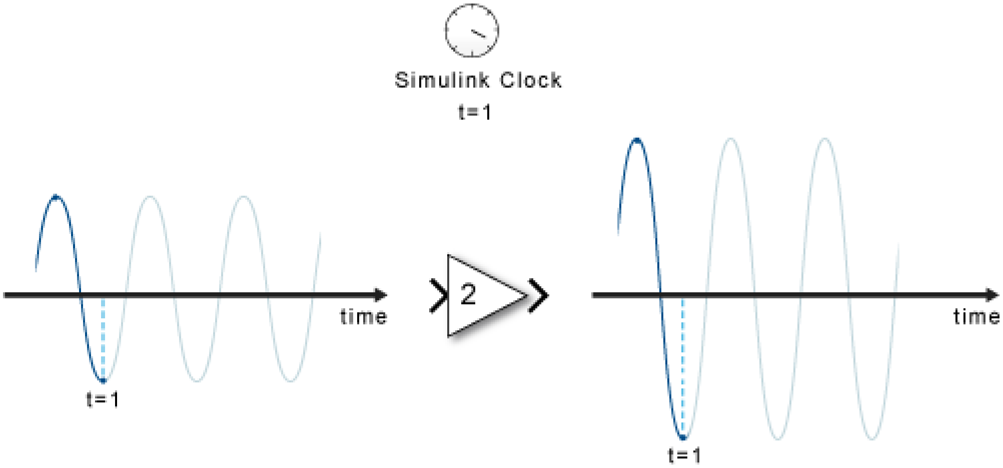


You can connect blocks to other blocks to represent more complex functionality and form systems. An audio player, for example, turns a digital file into sound: A digital representation is read from storage, gets interpreted mathematically, and is turned into sound physically. The software that processes the digital file to compute the sound waveform can be one block; the speaker that takes the waveform and turns it into sound can be another block. A component that generates the input is also a block in its own right.

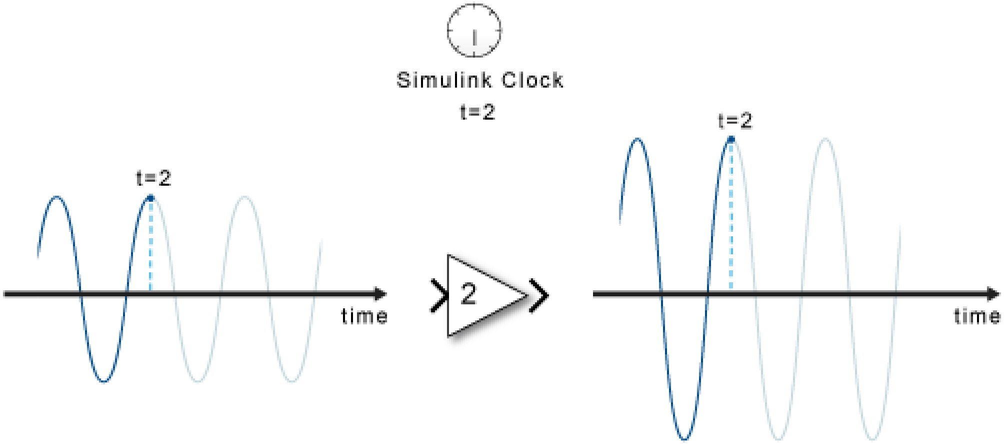
To model the sine wave input to the megaphone in Simulink, you would include a Sine Wave source:



The primary function of Simulink is to simulate behavior of system components over time. In its simplest form, this task involves keeping a clock, determining the order in which the blocks are to be simulated, and propagating the outputs, computed in the block diagram, to the next block. Consider the megaphone. At each time step, Simulink must compute the value of the sine wave, propagate it to the megaphone, and then compute the value of its output.



At each time step, each block computes its outputs from its inputs. Once all the signals in a diagram are computed at a given time step, Simulink determines the next time step (based on the model configuration and numerical solver algorithms) and advances the simulation clock. Then each block computes their output for this new time step.

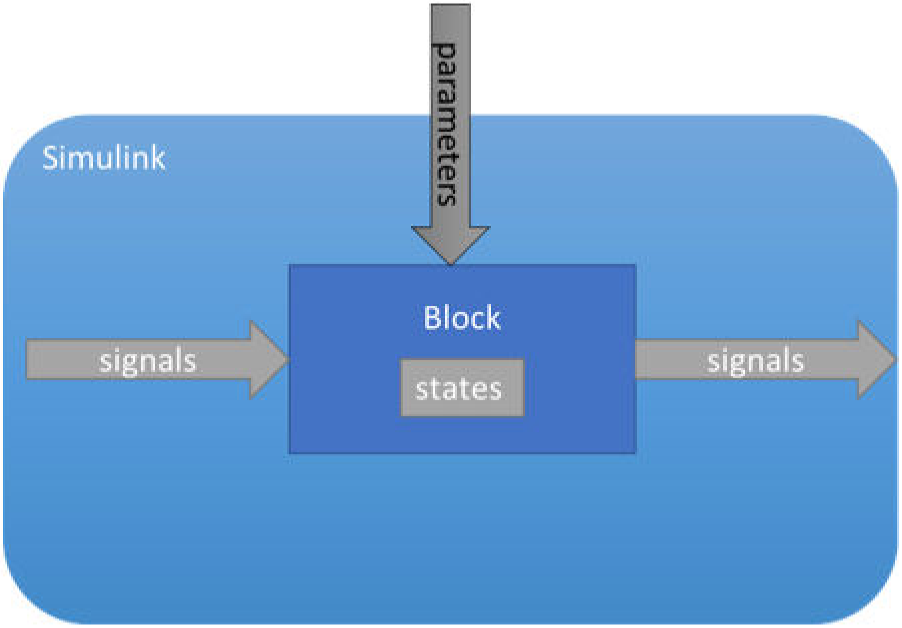


In simulation, time progresses differently from a real clock. Each time step takes as much time as it takes to finish the computations for that time step, whether that time step represents a fraction of a second or a few years.

Often, the effect of a component's input on its output is not instantaneous. For example, turning on a heater does not result in an instant change in temperature. Rather, this action provides input to a differential equation, and the history of the temperature (a *state*) is also a factor. When simulation requires solving a differential or difference equation, Simulink employs memory and numerical solvers to compute the state values for the time step.

Simulink handles data in three categories:

- Signals — Block inputs and outputs, computed during simulation
- States — Internal values, representing the dynamics of the block, computed during simulation
- Parameters — Values that affect the behavior of a block, controlled by the user



At each time step, Simulink computes new values for signals and states. By contrast, you specify parameters when you build the model and can occasionally change them while simulation is running.

3

Simple Simulink Model

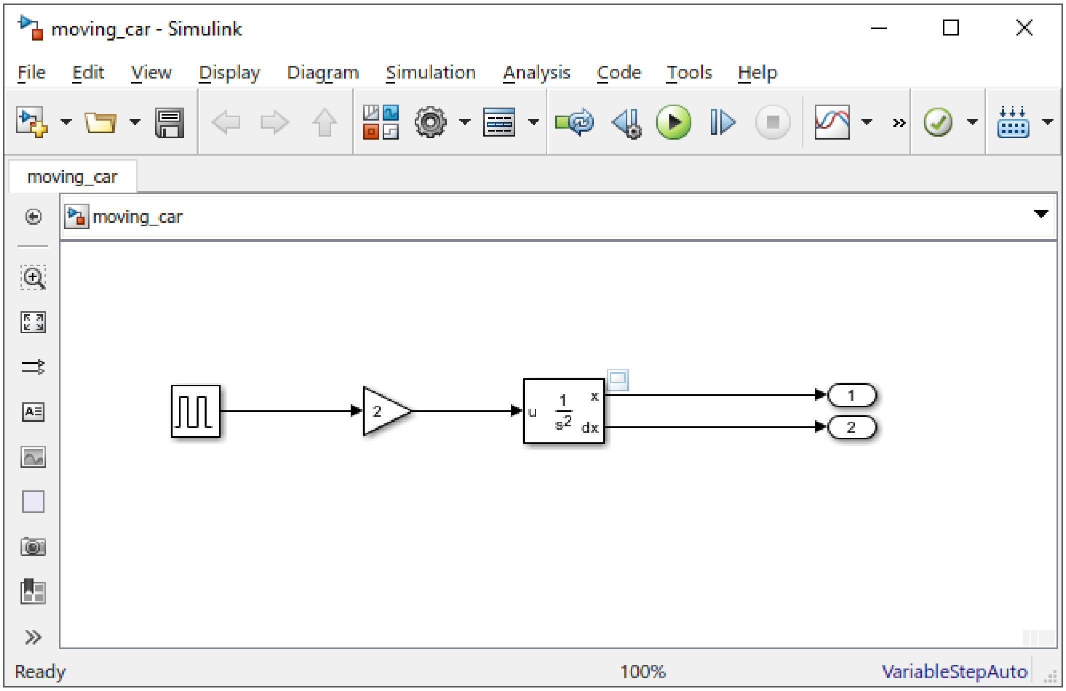
Create a Simple Model

In this section...

You can use Simulink to model a system and then simulate the dynamic behavior of that system. The basic techniques you use to create a simple model in this tutorial are the same as those you use for more complex models. This example simulates simplified motion of a car. A car is typically in motion while the gas pedal is pressed. After the pedal is released, the car idles and comes to a stop.

A Simulink block is a model element that defines a mathematical relationship between its input and output. To create this simple model, you need four Simulink blocks.

Block Name	Block Purpose	Model Purpose
Pulse Generator	Generate an input signal for the model	Represent the accelerator pedal
Gain	Multiply the input signal by a factor	Calculate how pressing the accelerator affects the car acceleration
Integrator, Second-Order	Integrate input signal twice	Obtain position from acceleration
Outport	Designate a signal as an output from the model	Designate the position as an output from the model

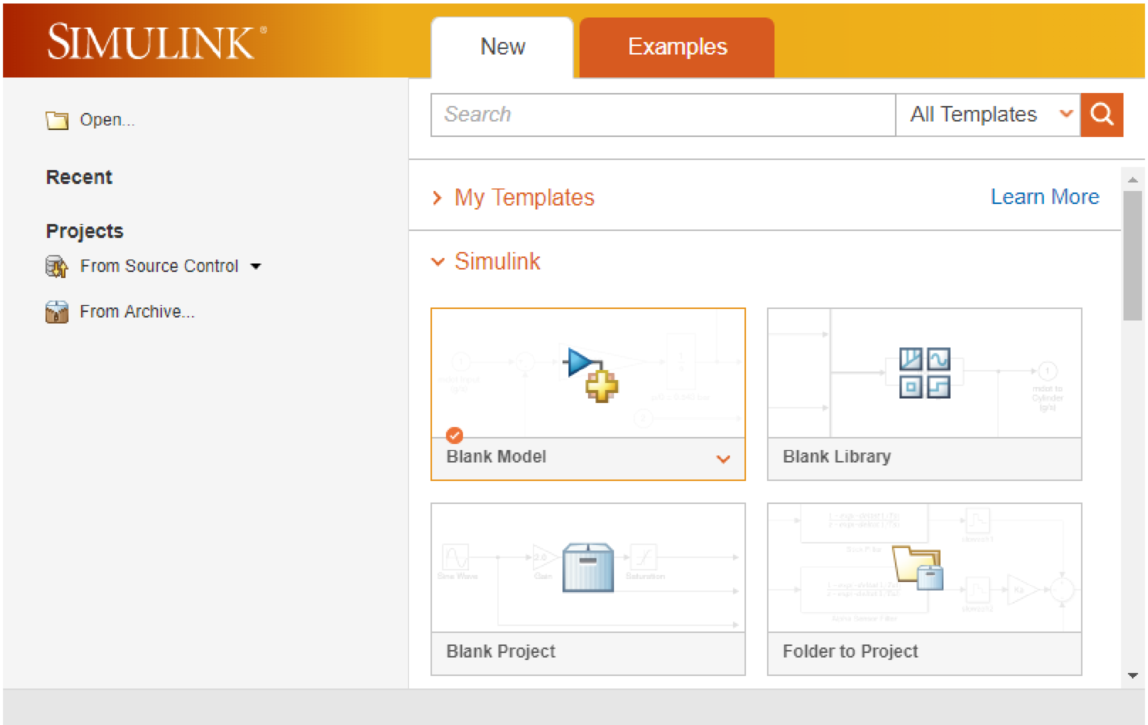


Simulating this model integrates a brief pulse twice to get a ramp. The results display in a Scope window. The input pulse represents a press of the gas pedal — 1 when the pedal is pressed and 0 when it is not. The output ramp is the increasing distance from the starting point.

Open New Model

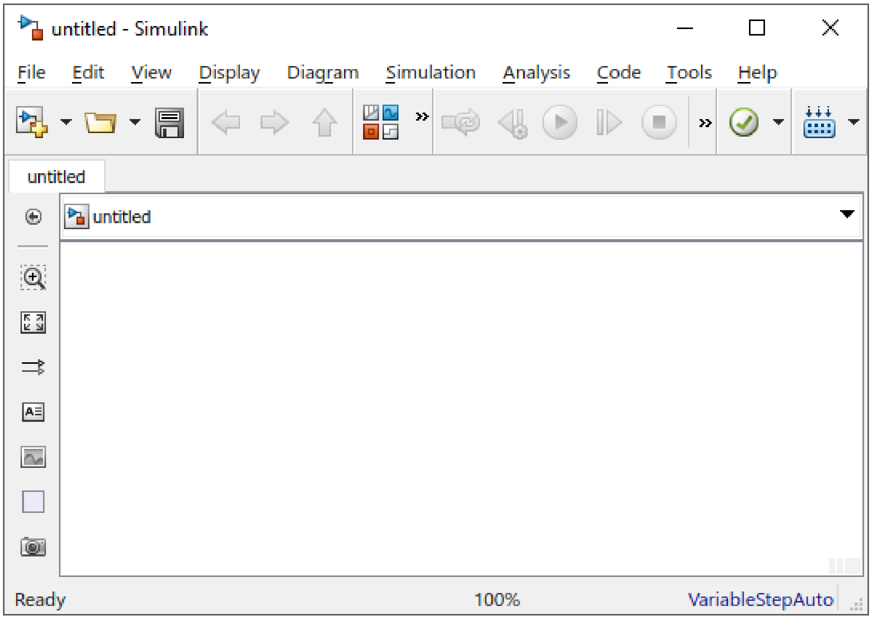
Use the Simulink Editor to build your models.

1 Start MATLAB. From the MATLAB toolstrip, click the **Simulink** button .



2 Click the **Blank Model** template.

The Simulink Editor opens.



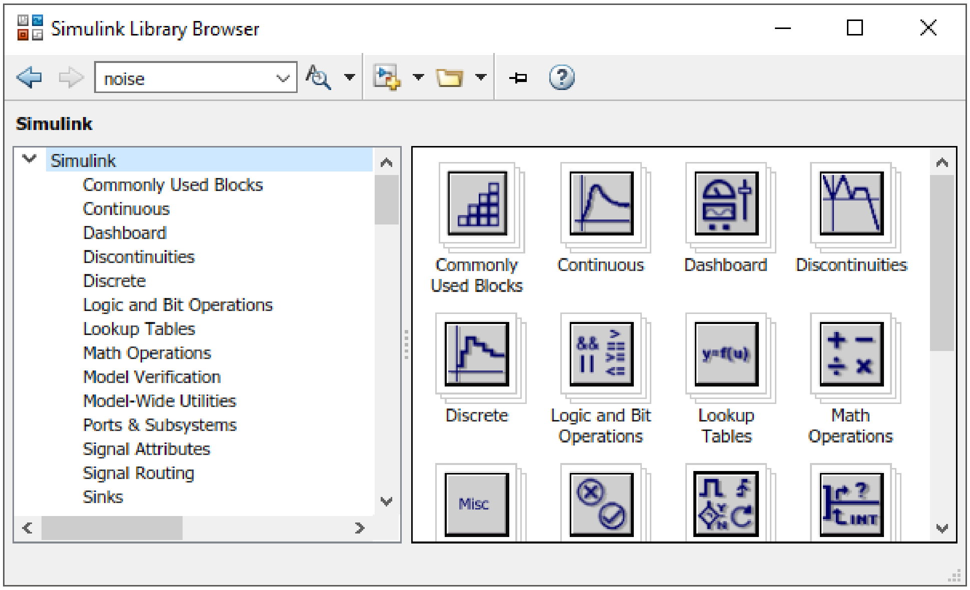
3 From the **File** menu, select **Save as**. In the **File name** text box, enter a name for your model, For example, `simple_model`. Click **Save**. The model is saved with the file extension `.slx`.

Open Simulink Library Browser

Simulink provides a set of block libraries, organized by functionality in the Library Browser. The following libraries are common to most workflows:

- Continuous — Blocks for systems with continuous states
- Discrete — Blocks for systems with discrete states
- Math Operations — Blocks that implement algebraic and logical equations
- Sinks — Blocks that store and show the signals that connect to them
- Sources — Blocks that generate the signal values that drive the model

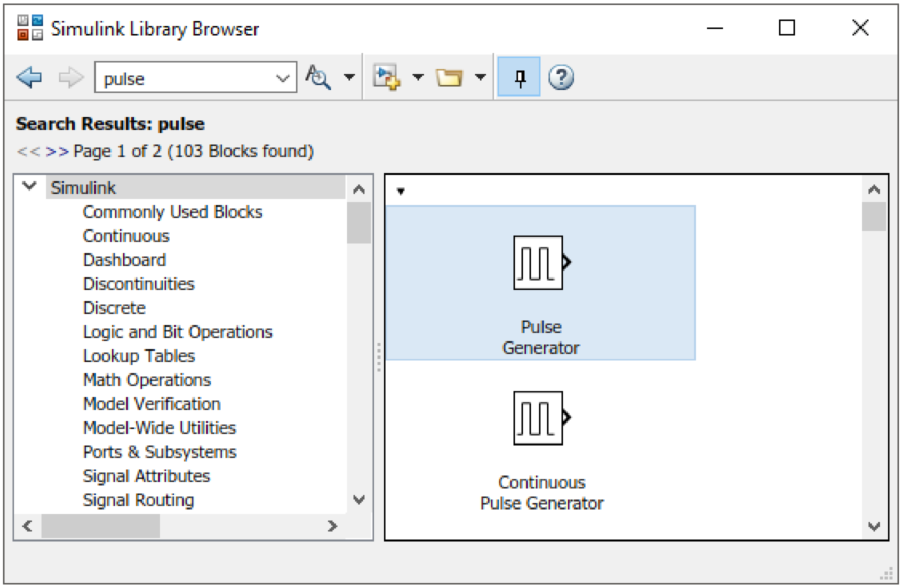
1 From the Simulink Editor toolbar, click the **Library Browser** button .



2 Set the Library Browser to stay on top of the other desktop windows. On the Library Browser toolbar, select the **Stay on top** button .

To browse through the block libraries, select a category and then a functional area in the left pane. To search all of the available block libraries, enter a search term.

For example, find the Pulse Generator block. In the search box on the browser toolbar, enter `pulse`, and then press the Enter key. Simulink searches the libraries for blocks with `pulse` in their name or description, and then displays the blocks.



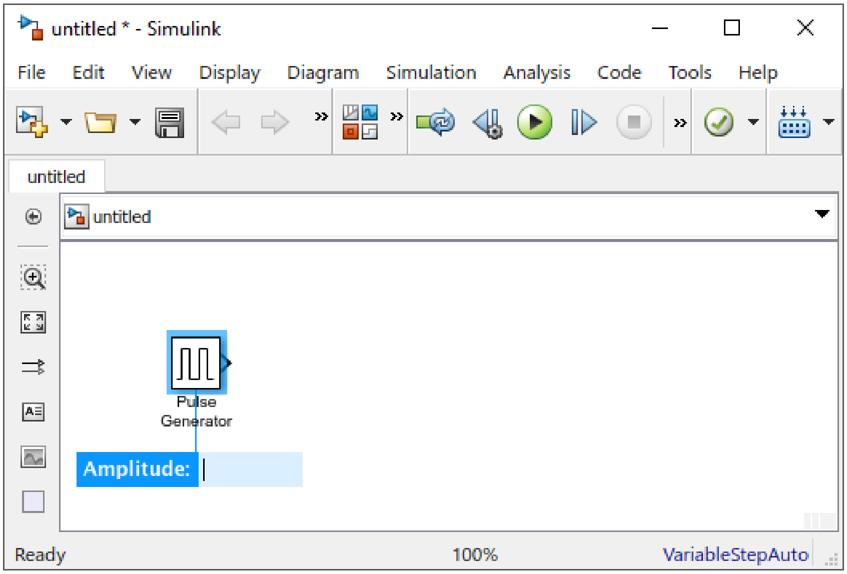
Get detailed information about a block. Right-click a block, and then select **Help for the Pulse Generator block**. The Help browser opens with the reference page for the block.

Blocks typically have several parameters. You can access all parameters by double-clicking the block.

Add Blocks to a Model

To start building the model, browse the library and add the blocks.

1 From the Sources library, drag the Pulse Generator block to the Simulink Editor. A copy of the Pulse Generator block appears in your model with a text box for the value of the **Amplitude** parameter. Enter 1.



Parameter values are held throughout the simulation.

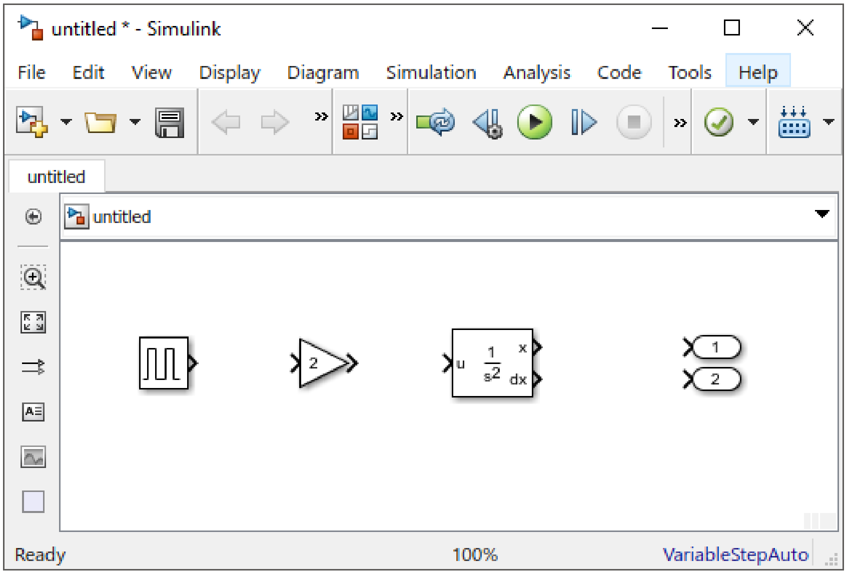
2 Add the following blocks to your model using the same approach.

Block	Library	Parameter
`Gain`	`Simulink/Math Operations`	Gain: 2
`Integrator, Second Order`	`Simulink/Continuous`	Initial condition: `0`
`Outport`	`Simulink/Sinks`	Port number: 1

Add a second Outport block by copying the existing one and pasting it at another point using keyboard shortcuts.

Your model now has the blocks you need.

3 Arrange the blocks as follows by clicking and dragging each block. To resize a block, click and drag a corner.

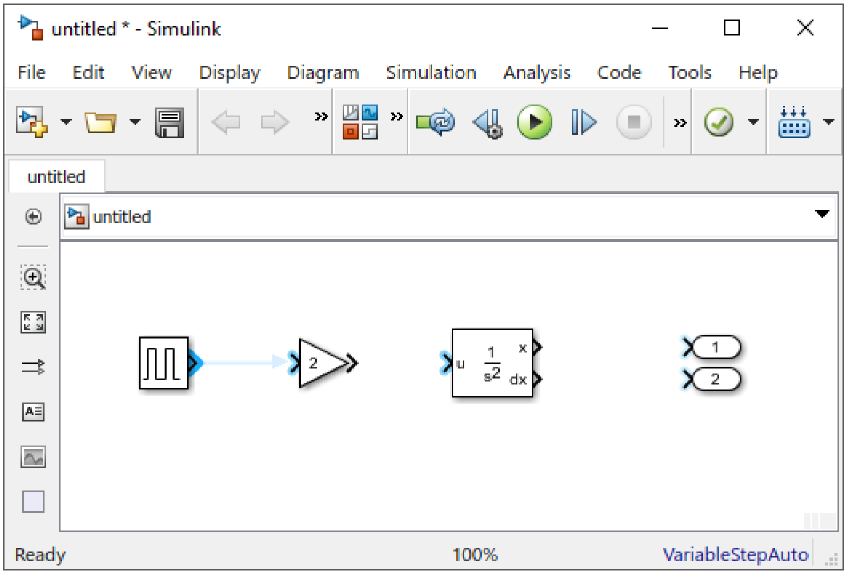


Connect Blocks

Connect the blocks by creating lines between output ports and input ports.

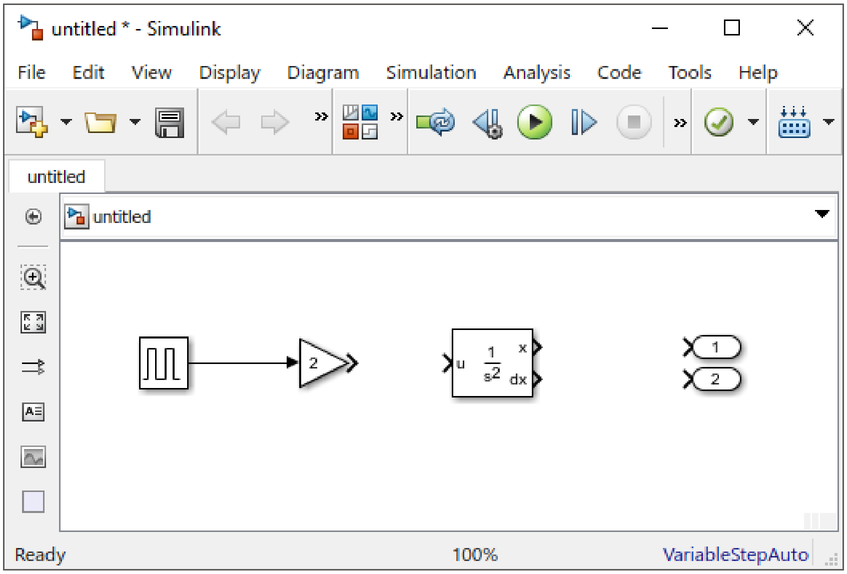
1 Click the output port on the right side of the Pulse Generator block.

The output port and all input ports suitable for a connection get highlighted.



2 Click the input port of the Gain block.

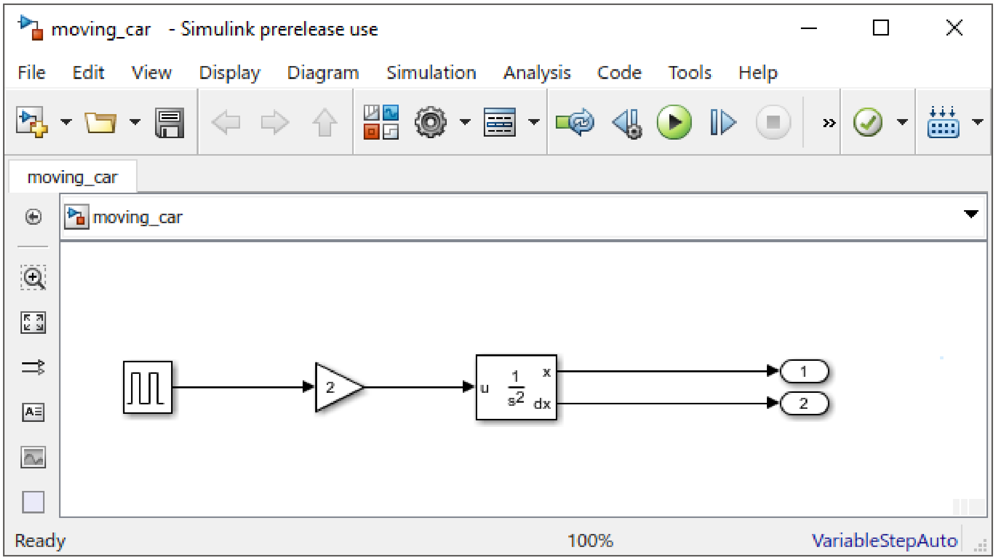
Simulink connects the blocks with a line and an arrow indicating the direction of signal flow.



3 Connect the output port of the Gain block to the input port on the Integrator, Second Order block.

4 Connect the two outputs of the Integrator, Second Order block to the two Outport blocks.

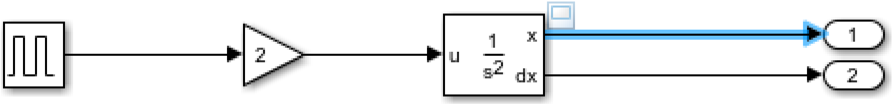
5 Save your model. Select **File > Save** and provide a name.



Add Signal Viewer

To view simulation results, connect the first output to a Signal Viewer.

Access the context menu by right-clicking the signal. Select **Create & Connect Viewer > Simulink > Scope**. A viewer icon appears on the signal and a scope window opens.

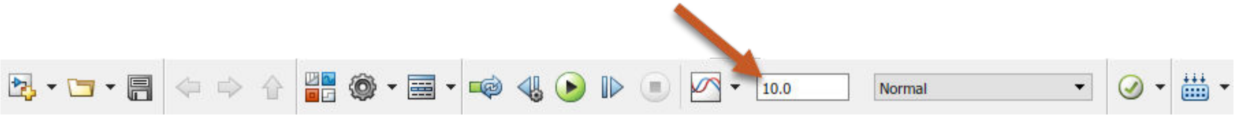


You can open the scope at any time by double-clicking the icon.

Run Simulation

After you define the configuration parameters, you are ready to simulate your model.

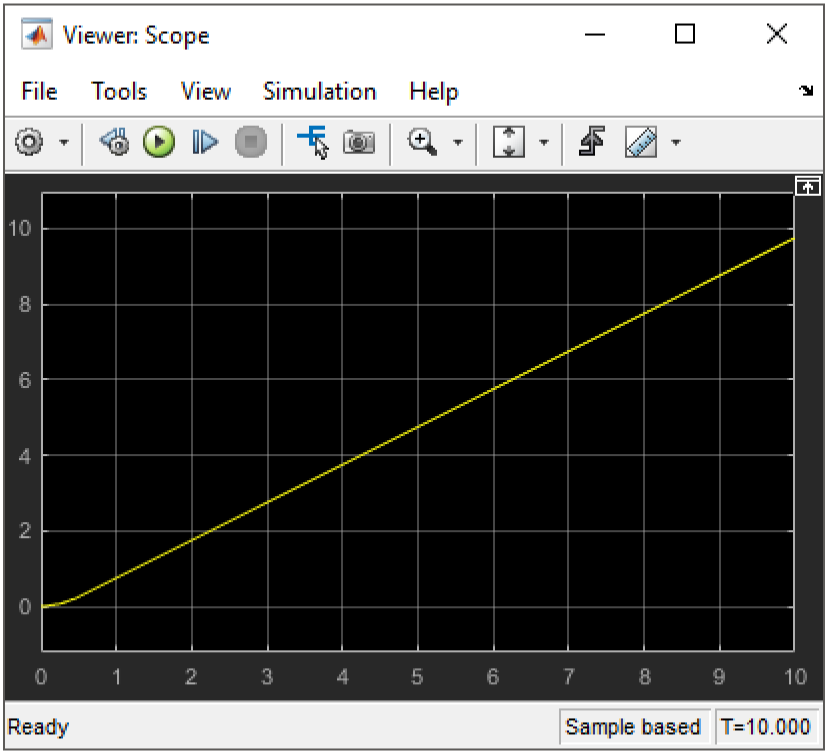
1 On the model window, set the simulation stop time by changing the value at the toolbar.



The default stop time of `10.0` is appropriate for this model. This time value has no unit. Time unit in Simulink depends on how the equations are constructed. This example simulates the simplified motion of a car for 10 seconds — other models could have time units in milliseconds or years.

2 To run the simulation, click the **Run** button ▶.

The simulation runs and produces the output in the viewer.



Refine Model

This example takes an existing model, `moving_car.slx`, and models a proximity sensor based on this motion model. In this scenario, a digital sensor measures the distance between the car an obstacle 10 m (30 ft) away. The model outputs the sensor measurement, and the position of the car, taking these conditions into consideration:

- The car comes to a hard stop when it reaches the obstacle.
- In the physical world, a sensor measures the distance imprecisely, causing random numerical errors.
- A digital sensor operates at fixed time intervals.

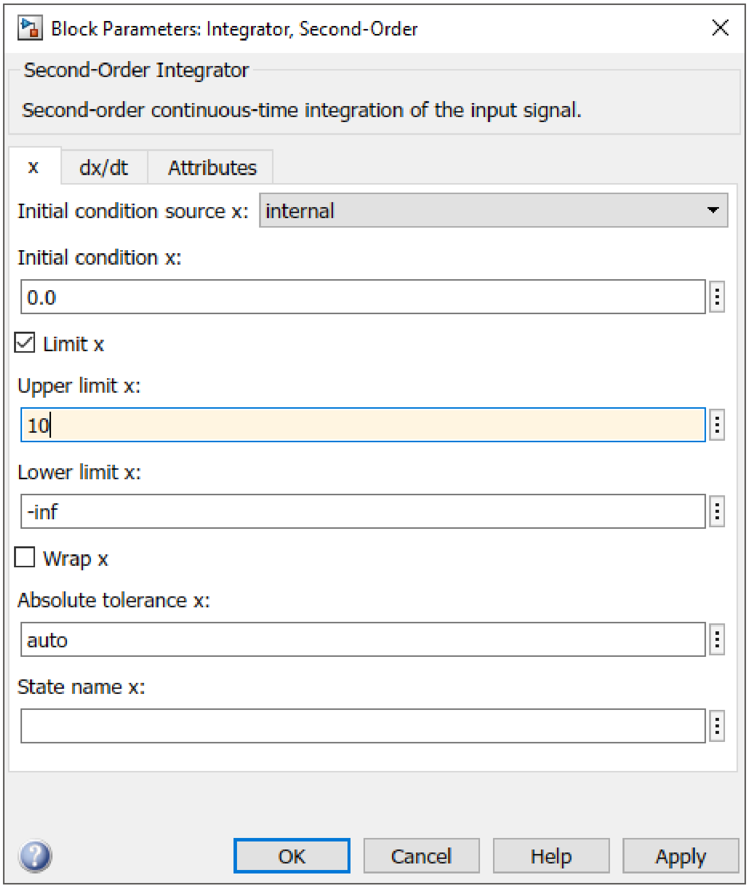
Change Block Parameters

To start, open the `moving_car` model. In the MATLAB Command Window, enter

```
open_system(fullfile(matlabroot,...
'help', 'toolbox', 'simulink', 'examples', 'moving_car'))
```

You first need to model the hard stop when the car position reaches `10`. The Integrator, Second Order block has a parameter for that purpose.

1. Double-click the Integrator, Second Order block. The Block Parameters dialog box appears.
2. Select **Limit x** and enter `10` for **Upper limit x**.



Block Parameters: Integrator, Second-Order

Second-Order Integrator

Second-order continuous-time integration of the input signal.

Initial condition source x: internal

Initial condition x: 0.0

☑ Limit x

Upper limit x: 10

Lower limit x: -inf

□ Wrap x

Absolute tolerance x: auto

State name x:

The background color for the parameter changes to indicate a modification that is not applied to the model.

3 Click **OK** to apply the changes and close the dialog box.

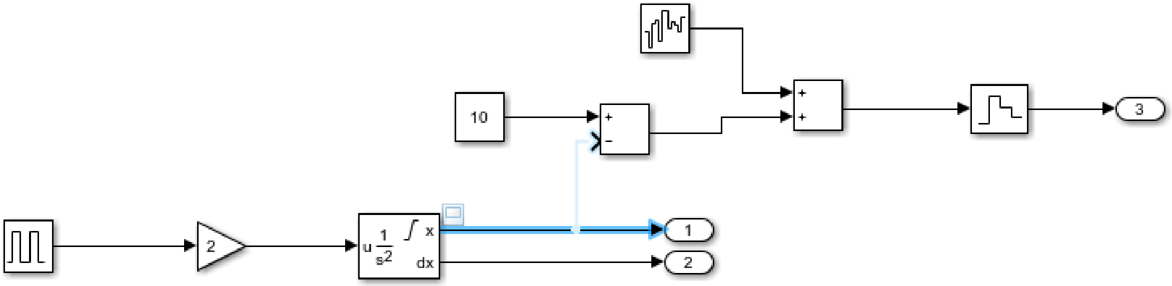
Add New Blocks and Connections

Add a sensor that measures the distance from the obstacle.

1 Modify the model. Extend the model window to accommodate the new blocks as necessary.

- Find the actual distance. To find the distance between the obstacle position and the vehicle position, add the Subtract block. Also add the Constant block to set the constant value of `10` for the position of the obstacle.
- Model the imperfect measurement that would be typical to a real sensor. Generate noise by using the Band-Limited White Noise block from the Sources library. Set the **Noise power** parameter to `0.001`. Add the noise to the measurement by using an Add block from the Math Operations library.
- Model the digital sensor that fires every 0.1 seconds. In Simulink, sampling of a signal at a given interval requires a sample and hold, implemented by a zero-order hold. Add the Zero-Order Hold block from the Discrete library. After you add the block to the model, change the **Sample Time** parameter to `0.1`.
- Add another Outport to connect to the sensor output. Leave the **Port number** parameter as default.

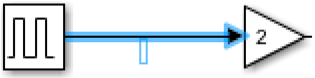
2 Connect the new blocks. Note that the output of the Integrator, Second-Order block is already connected to another port. To create a branch in that signal, left-click the signal to highlight potential ports for connection, and click the appropriate port.



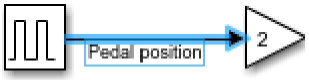
Annotate signals

Add signal names to the model to make it easier to understand.

1 Double-click the signal. An editable textbox appears.

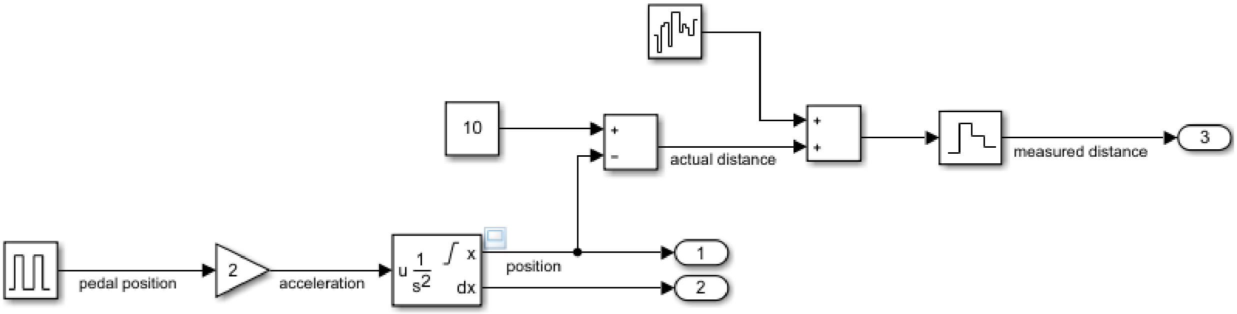


2 Type the signal name.



3 To finish, click away from the textbox.

4 Repeat these steps to add the names as shown.

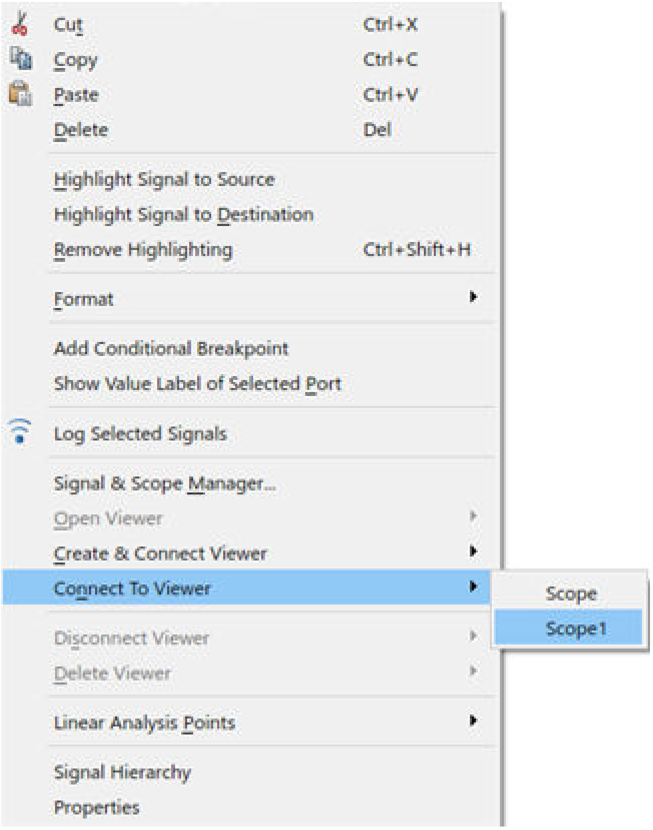


Compare Multiple Signals

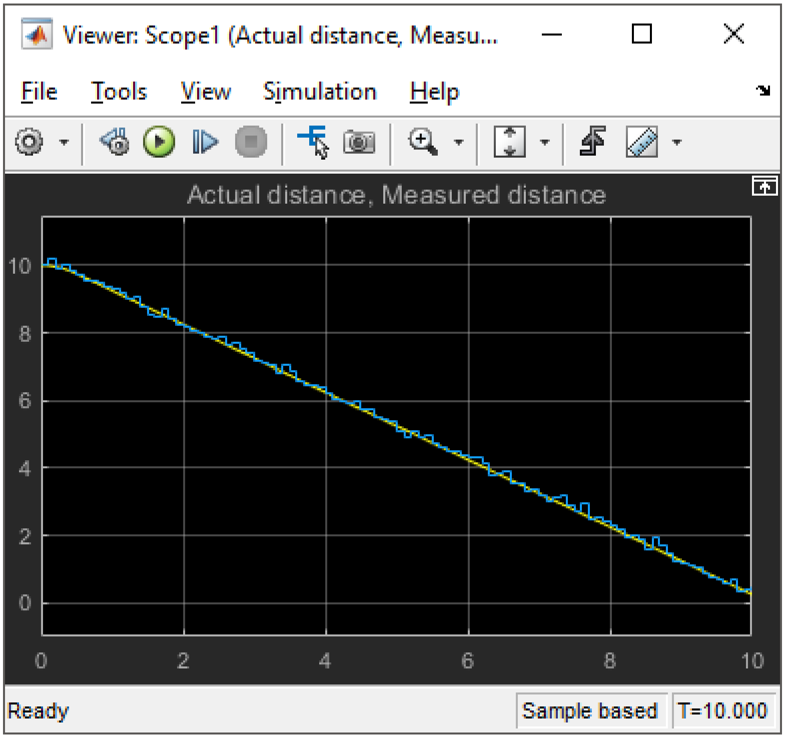
Compare the *actual distance* signal with the *measured distance* signal.

1 Create and connect a Scope to the *actual distance*. Note that the name of the signal appears in the viewer title.

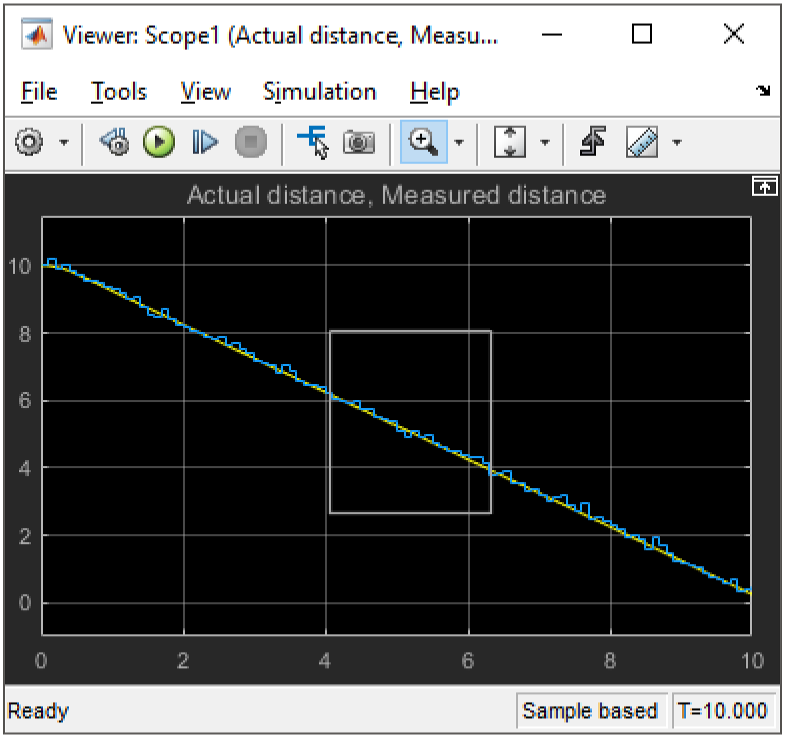
2 Add the *measured distance* signal to the same viewer. Right-click the signal, and select **Connect to Viewer > Scope1**. Make sure you are connecting to the viewer you created in the previous step.



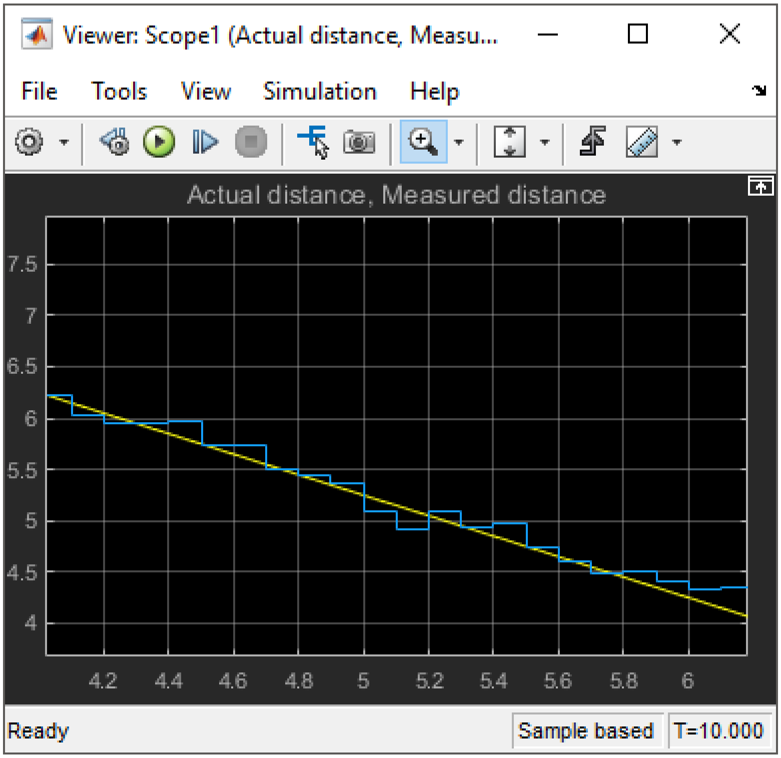
3 Run the model. The Viewer shows the two signals, *actual distance* in yellow and *measured distance* in blue.


Viewer: Scope1 (Actual distance, Measu...
File
Tools
View
Simulation
Help
Actual distance, Measured distance
10
8
6
4
2
0
0
2
4
6
8
10
Ready
Sample based
T=10.000

4 Zoom into the graph to observe the effect of noise and sampling. Click the **Zoom** button . Left-click and drag a window around the region you want to see.


Viewer: Scope1 (Actual distance, Measu...
File
Tools
View
Simulation
Help
Actual distance, Measured distance
10
8
6
4
2
0
0
2
4
6
8
10
Ready
Sample based
T=10.000

You can repeatedly zoom in to observe the details.



From the plot, you can see that the measurement can deviate from the actual value by as much as 0.3 m. This information becomes useful when designing a safety feature, for example, a collision warning.

See Also

Blocks
Add | Band-Limited White Noise | Constant | Gain | Pulse Generator | Second-Order Integrator, Second-Order Integrator Limited | Zero-Order Hold

Related Examples

- “Model and Validate a System” on page 1-16

4

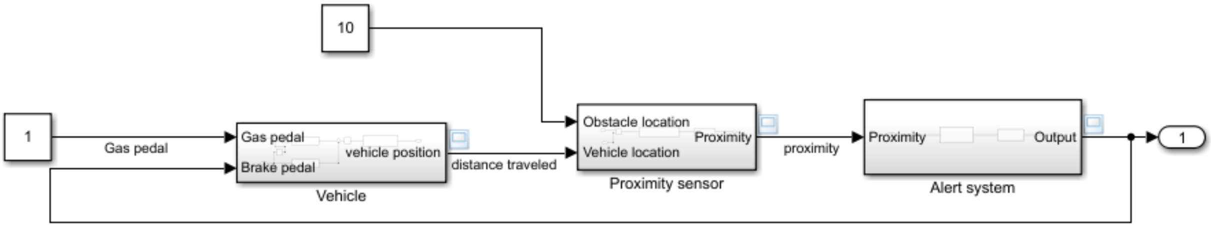
Navigate a Simulink Model

Navigate Model

Simulink models are hierarchical, so you can build models using both top-down and bottom-up approaches. You can view the system at a high level, then drill down to see increasing levels of model detail. This approach provides insight into how a model is organized and how parts interact.

To start, open the `smart_braking` model. In the MATLAB Command Window, enter

```
open_system(fullfile(matlabroot,...
'help', 'toolbox', 'simulink', 'examples', 'smart_braking'))
```



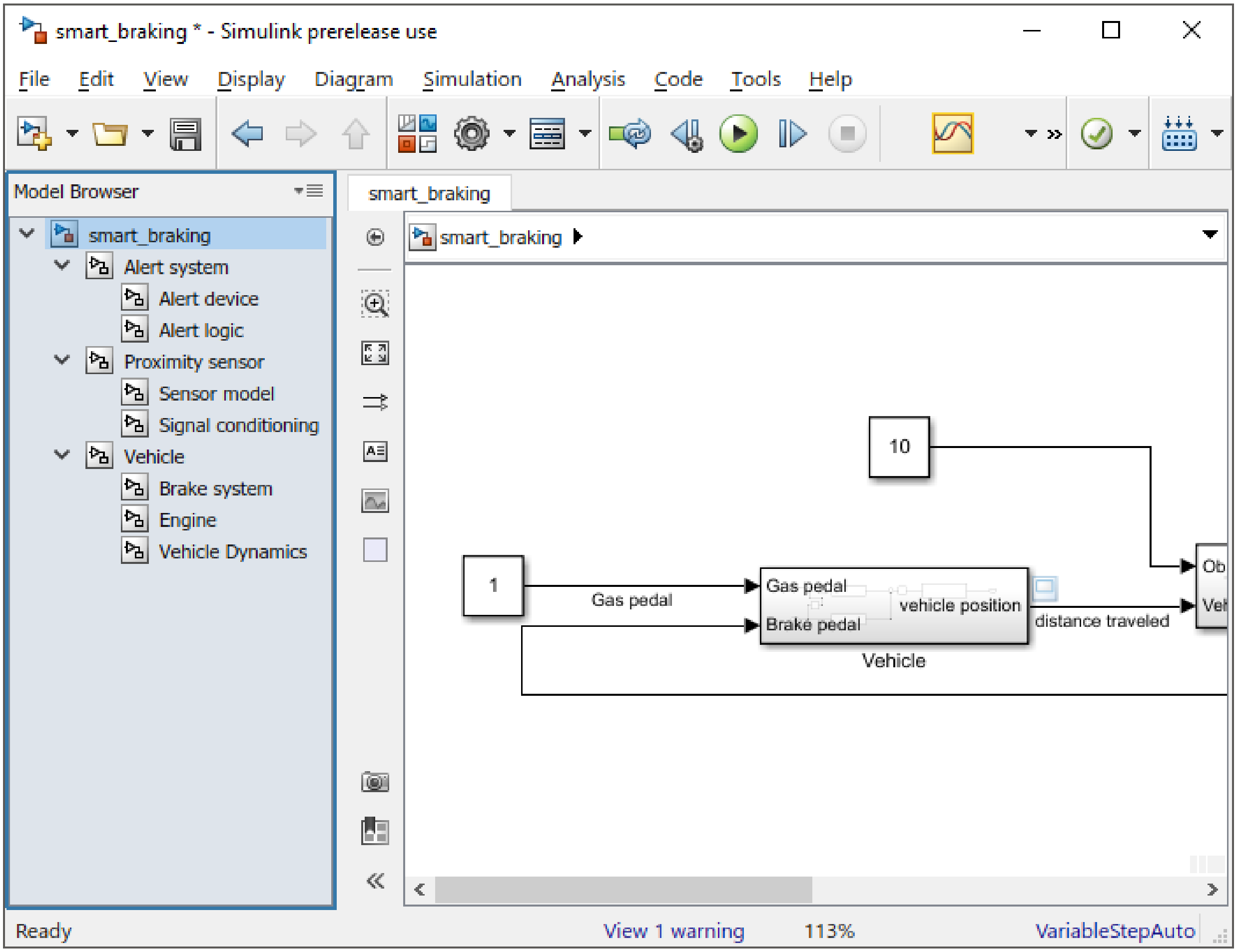
This model includes the following components and data flow:

- A vehicle moves as the gas pedal is pressed.
- A proximity sensor measures its distance from an obstacle.
- An alert system generates an alarm based on that proximity.
- The alarm automatically controls the brake to avoid hitting the obstacle.

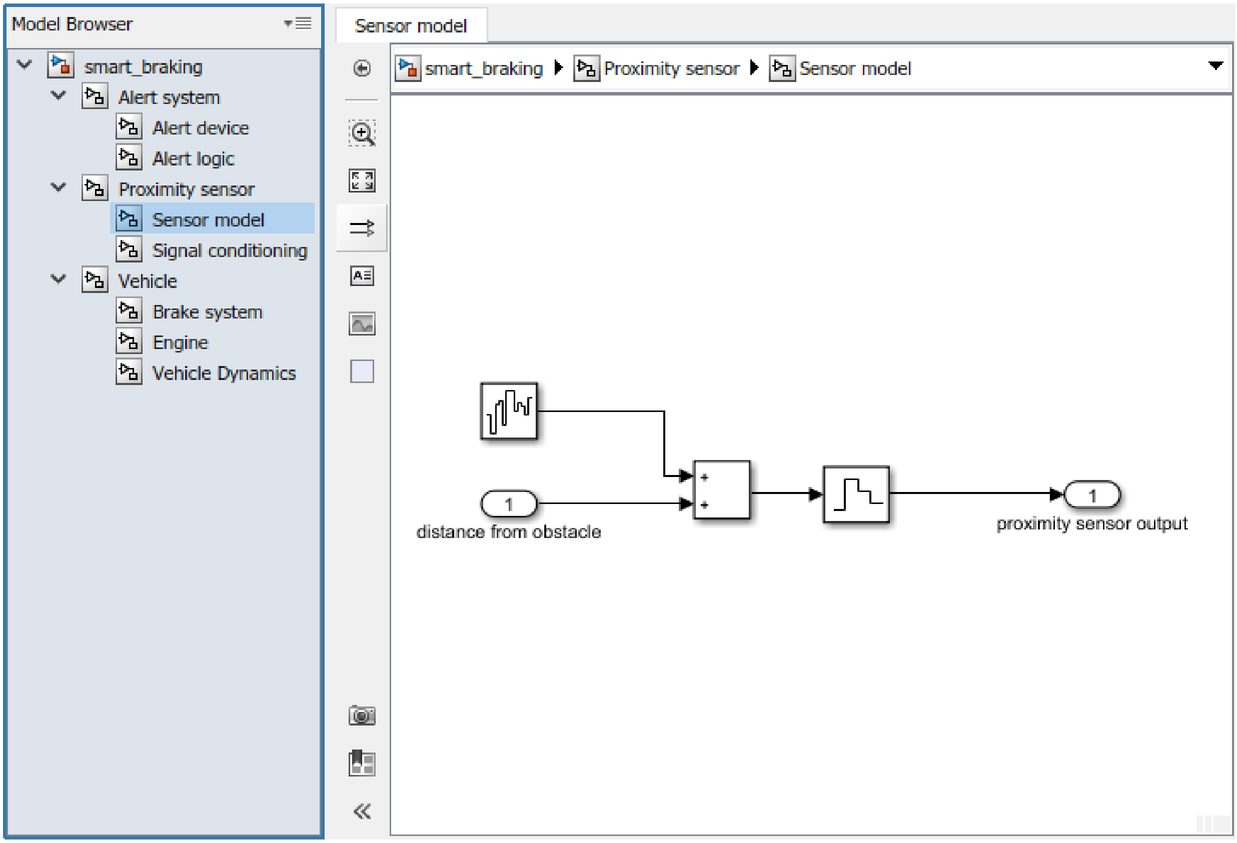
Navigate Through Model Hierarchy

You connect blocks together to model complex components. In this model, Vehicle, Proximity sensor, and Alert system are all complex components with multiple blocks, and they exist in a hierarchy of subsystems. To view its contents, double-click any subsystem:

To view the complete tree, click the **Hide/Show Model Browser** button » at the bottom left corner of the model window.



The Model Browser shows that all subsystems you view at the top level also have subsystems of their own. Click **>** icons to see the subsystems. You can navigate through the hierarchy in the Model Browser. For example, click the Sensor model subsystem:



Observe that the subsystem is highlighted in the Model Browser. The address bar also shows which subsystem you are viewing. To open the subsystem in a separate window instead, right-click the subsystem name and select **Open In New Window**.

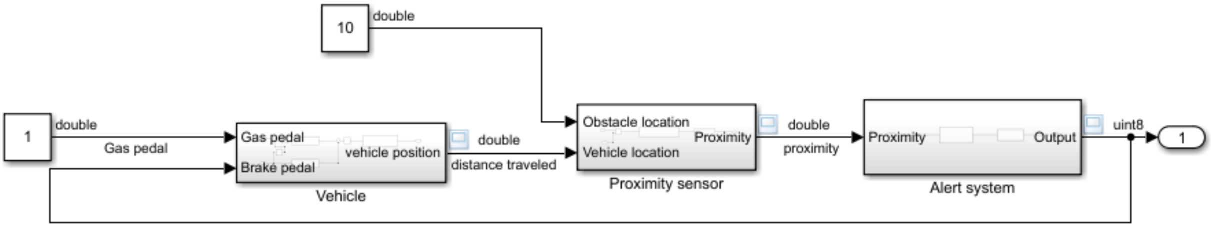
Every input or output port on a subsystem has a corresponding Inport or Outport block inside the subsystem. These blocks indicate data transfer between a subsystem and its parent. In the case of multiple inputs or outputs, the number on the block designates which port it connects to on the subsystem.

View Signal Attributes

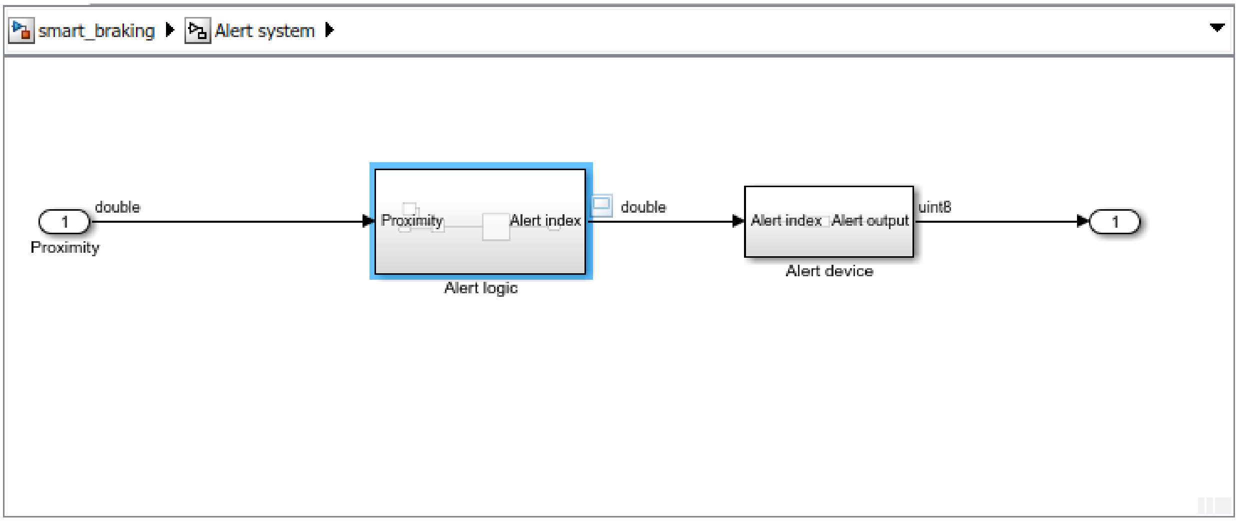
The signal lines in Simulink indicate data transfer from block to block. These signals have attributes essential to the function of the model:

- Size: Scalar, vector, or matrix
- Data type: String, double, unsigned integer, etc.
- Sample time: The fixed time interval at which this signal has an updated value, or continuous sampling

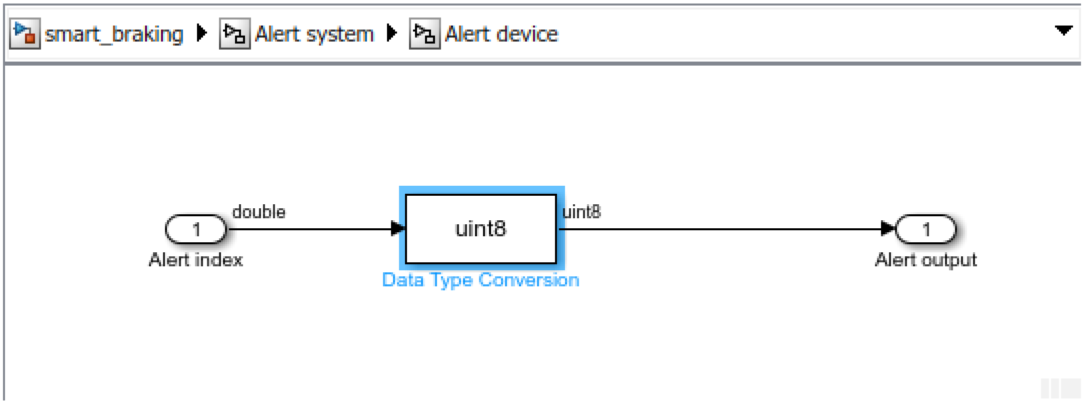
To show the data type of all signals on a model, select **Display > Signals & Ports > Port Data Types**:



The model displays data types along the signal types. Observe that most signals are double, except the output of the Alert subsystem. Double-click this subsystem to investigate why:

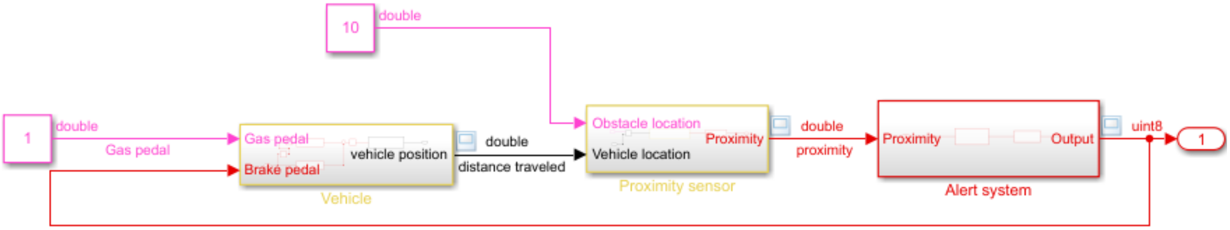


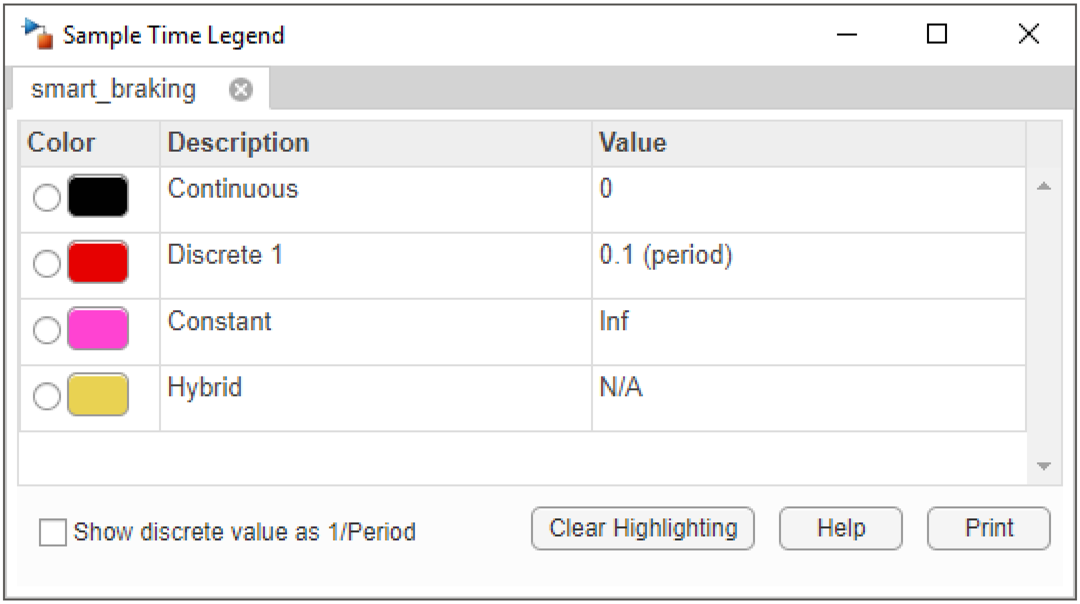
The labels in this subsystem suggests that data type change occurs in the Alert device subsystem, double-click to investigate:



This shows that the Alert device component converts the alert index signal from a double to an integer. You can set the data type at sources, or use a Data Type Conversion block from the Signal Attributes library. The double data type, the default, provides the best numerical precision and is supported in all blocks. It also uses the most memory and computing power. Other numerical data types serve to model typical embedded systems where memory and computing power are limited.

To show sample times, select **Display > Sample Time > Colors**. This updates the model to show different colors for different sample times, and also displays a legend:





Sample Time Legend

smart_braking

Color	Description	Value
○ ⬛	Continuous	0
○ 🟥	Discrete 1	0.1 (period)
○ 🟪	Constant	Inf
○ 🟨	Hybrid	N/A

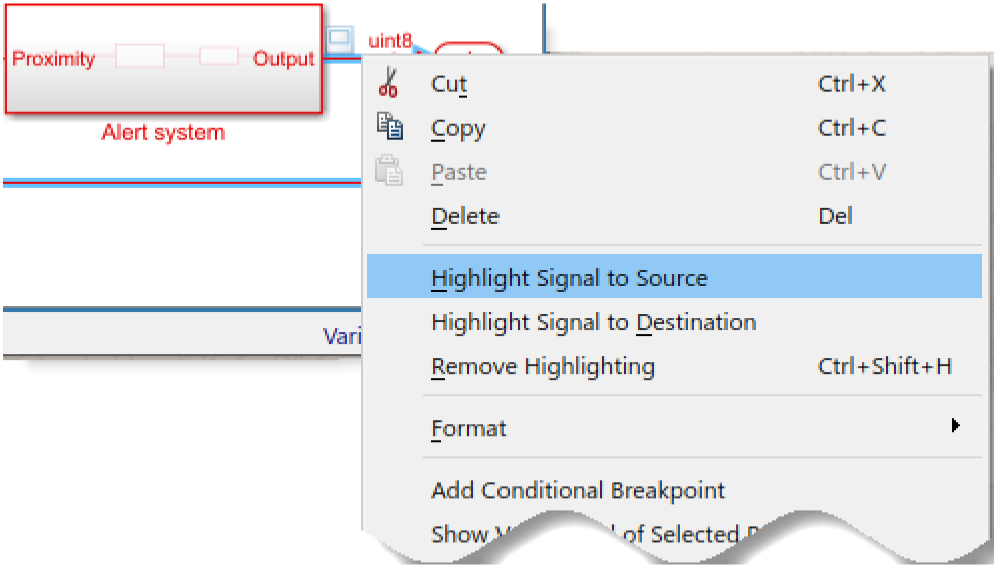
- A block or signal with continuous dynamics is black. They update as often as Simulink requires to make the computations as close to the physical world as possible.
- A block or signal that is constant is magenta. They remain unchanged through simulation.
- A discrete block or signal that updates at the lowest fixed interval is red: They update only at fixed intervals. If the model contains components with different fixed sample times, each sample time has a different color.
- A subsystem that contains continuous and discrete components are yellow: They are hybrid systems.

Trace a Signal

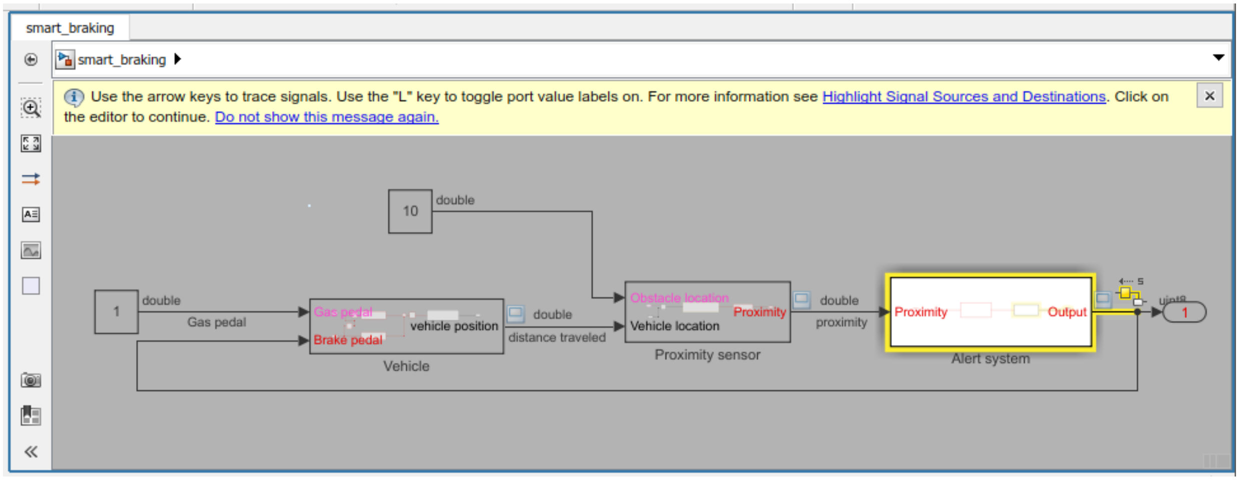
This model has a constant source and a discrete output. To determine where the sampling scheme changes., trace the output signal through blocks:

1 Open the Model Browser: Click the **Hide/Show Model Browser** button ».

2 Highlight the source of the output signal: Right-click the signal and select **Highlight Signal to Source**.

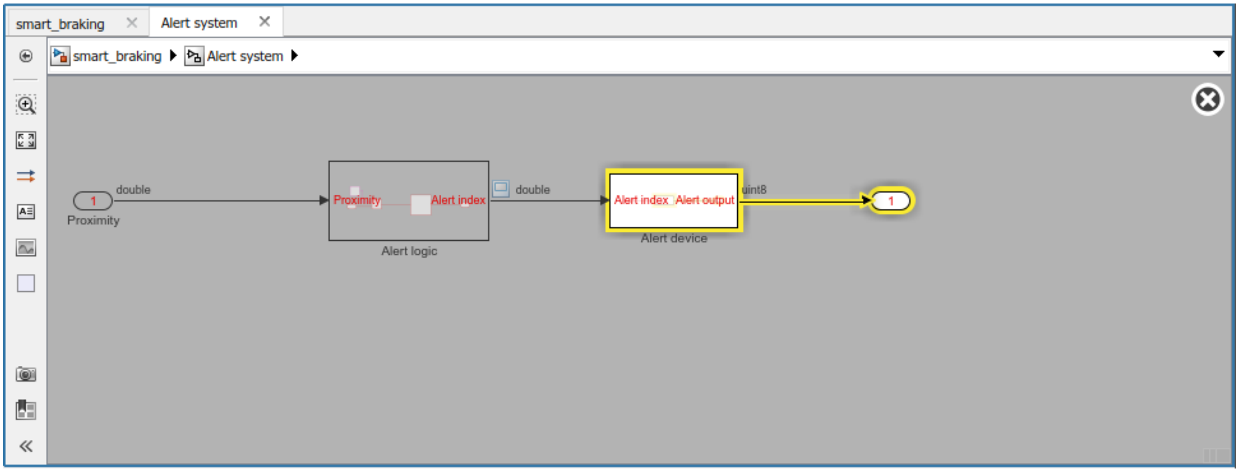

Proximity
Output
Alert system
uint8
Vari
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Highlight Signal to Source
Highlight Signal to Destination
Remove Highlighting Ctrl+Shift+H
Format
Add Conditional Breakpoint
Show V of Selected P

This takes the editor into highlight mode. Click the editor to continue. Make sure there is a blue frame around the editor.

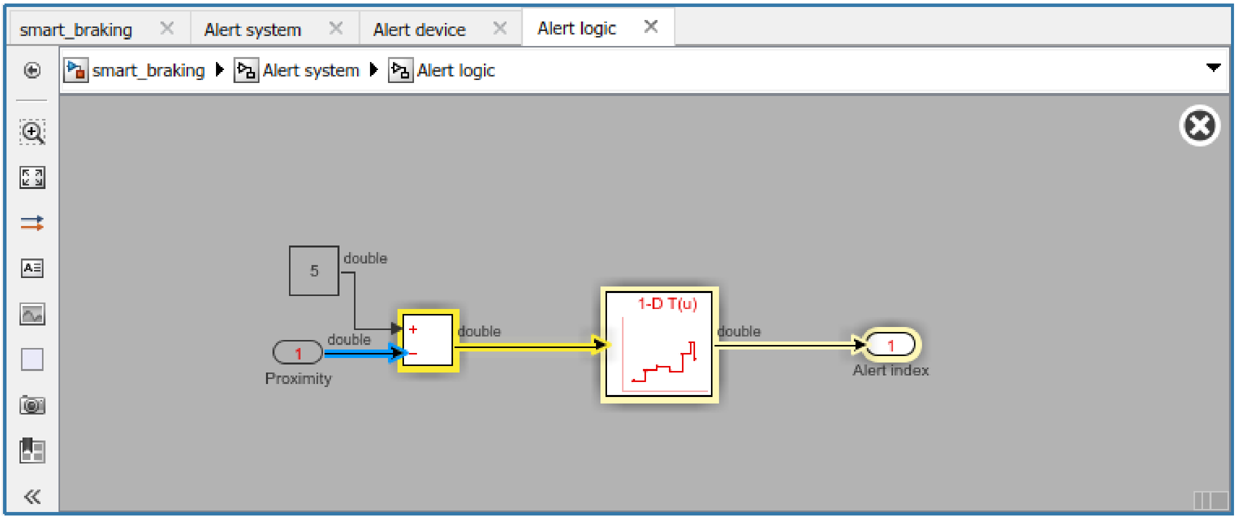


Use the arrow keys to trace signals. Use the "L" key to toggle port value labels on. For more information see Highlight Signal Sources and Destinations. Click on the editor to continue. Do not show this message again.

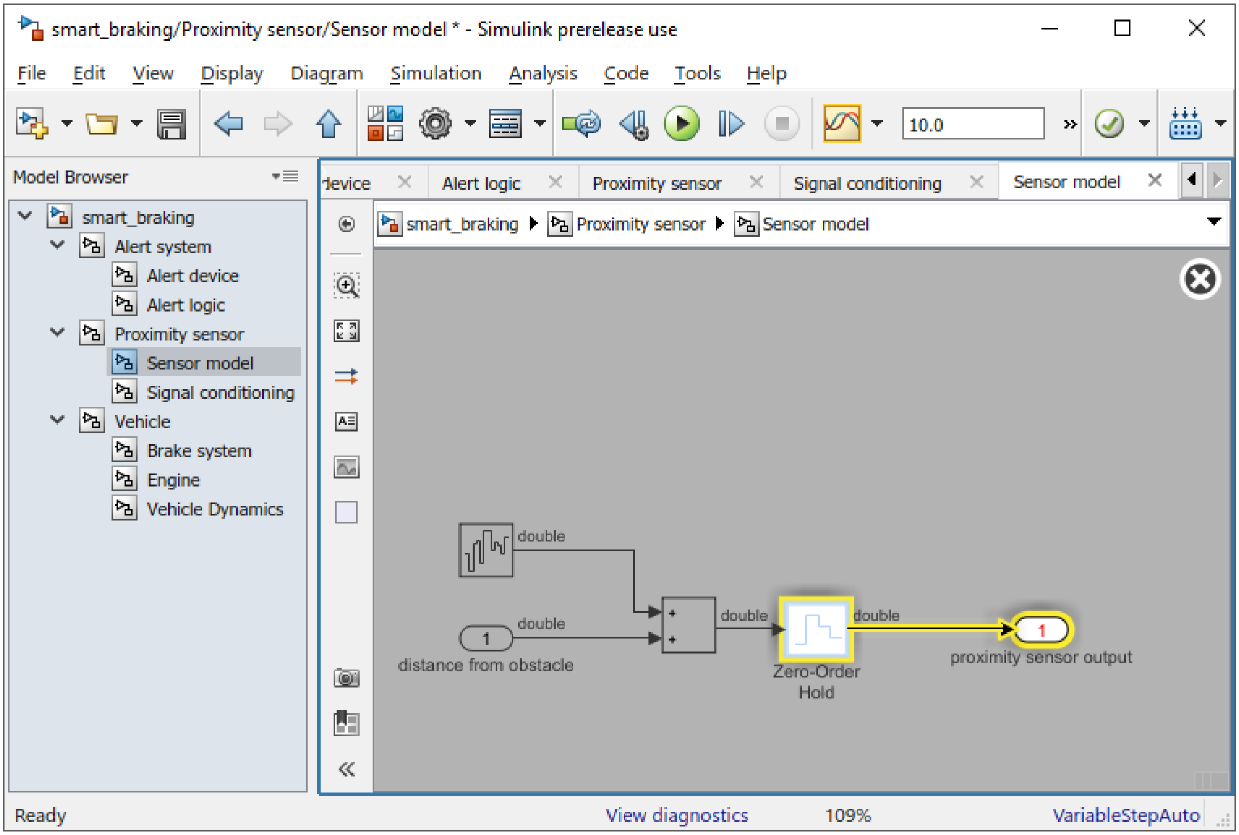
3 To continue tracing the signal to its source, press the left arrow key.



4 Keep tracing the signal to its source until you reach the Alert logic subsystem. You see that the Subtract block has two inputs. Choose the signal path from the Inport by pressing the down arrow key.



5 To find the source of the discretization, keep pressing the left arrow and note the colors of port names that reflect the sample time.



You find that the Zero-Order Hold block in the Sensor model subsystem does the conversion from continuous to discrete.

Simulation of a Bouncing Ball

This example shows how to use two different approaches to modeling a bouncing ball using Simulink®.

Overview

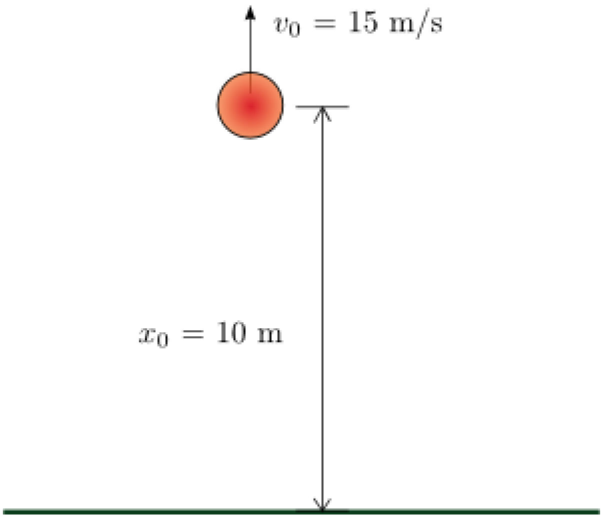


Figure 1: A ball is thrown up with a velocity of 15 m/s from a height of 10 m.

A bouncing ball model is a classic example of a **hybrid dynamic system**. A hybrid dynamic system is a system that involves both continuous dynamics, as well as, discrete transitions where the system dynamics can change and the state values can jump. The continuous dynamics of a bouncing ball is simply given by:

$$\frac{dv}{dt} = -g,$$

$$\frac{dx}{dt} = v,$$

where g is the acceleration due to gravity, $x(t)$ is the position of the ball and $v(t)$ is the velocity. Therefore, the system has two continuous states: position x and velocity v.

The hybrid system aspect of the model originates from the modeling of a collision of the ball with the ground. If one assumes a partially elastic collision with the ground, then the velocity before the collision, v^-, and velocity after the collision, v^+, can be related by the coefficient of restitution of the ball, κ, as follows:

$$v^+ = -\kappa v^-, \qquad x = 0$$

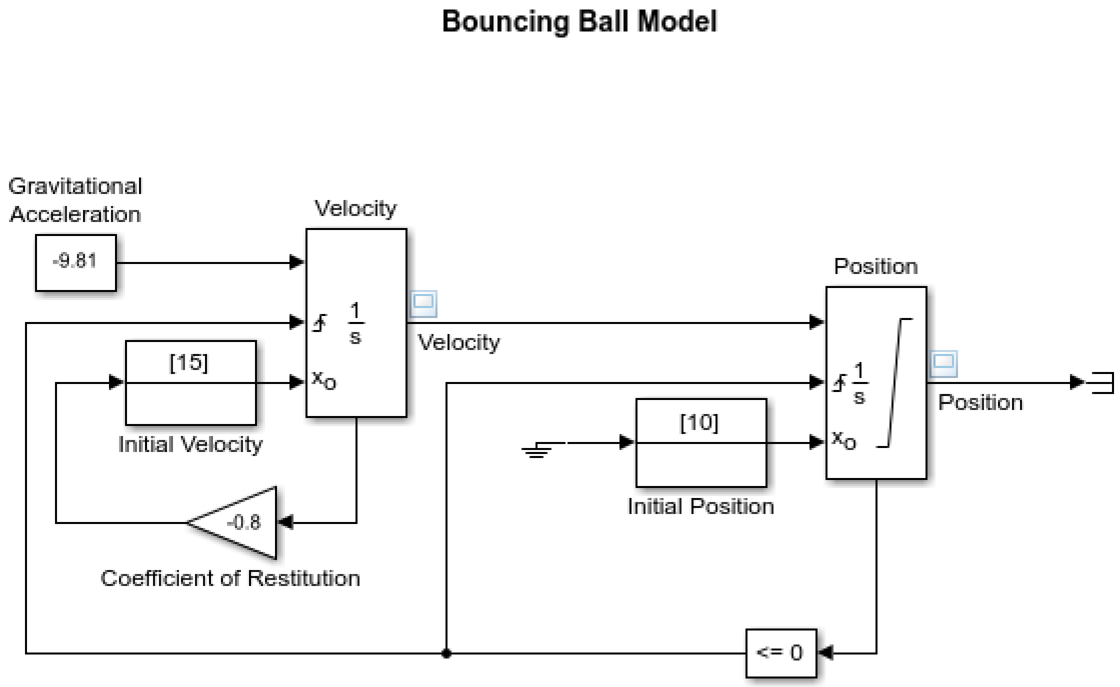
The bouncing ball therefore displays a jump in a continuous state (velocity) at the transition condition, $x = 0$.

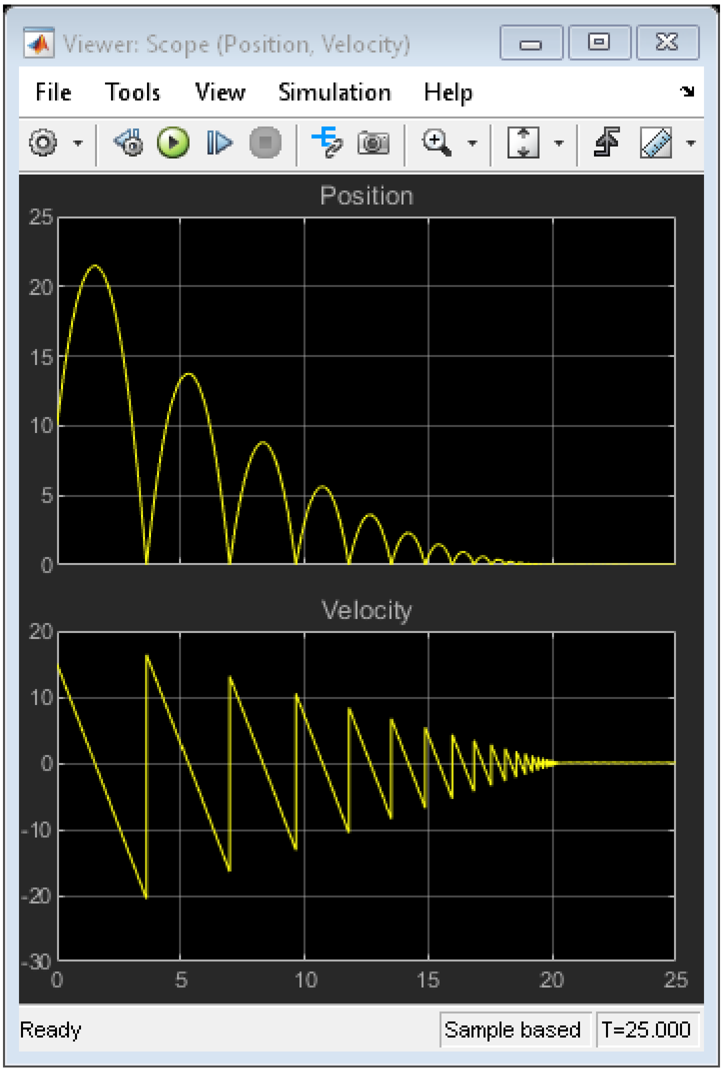
A bouncing ball is one of the simplest models that shows the **Zeno phenomenon**. Zeno behavior is informally characterized by an infinite number of events occurring in a finite time interval for certain hybrid systems. As the ball loses energy in the bouncing ball model, a large number of collisions with the ground start occurring in successively smaller intervals of time. Hence the model experiences Zeno behavior. Models with **Zeno behavior** are inherently difficult to simulate on a computer, but are encountered in many common and important engineering applications.

Using Two Integrator Blocks to Model a Bouncing Ball

You can use two Integrator blocks to model a bouncing ball. The Integrator on the left is the velocity integrator modeling the first equation and the Integrator on the right is the position integrator. Navigate to the position integrator block dialog and observe that it has a lower limit of zero. This condition represents the constraint that the ball cannot go below the ground.

The state port of the position integrator and the corresponding comparison result is used to detect when the ball hits the ground and to reset both integrators. The state port of the velocity integrator is used for the calculation of v^+.



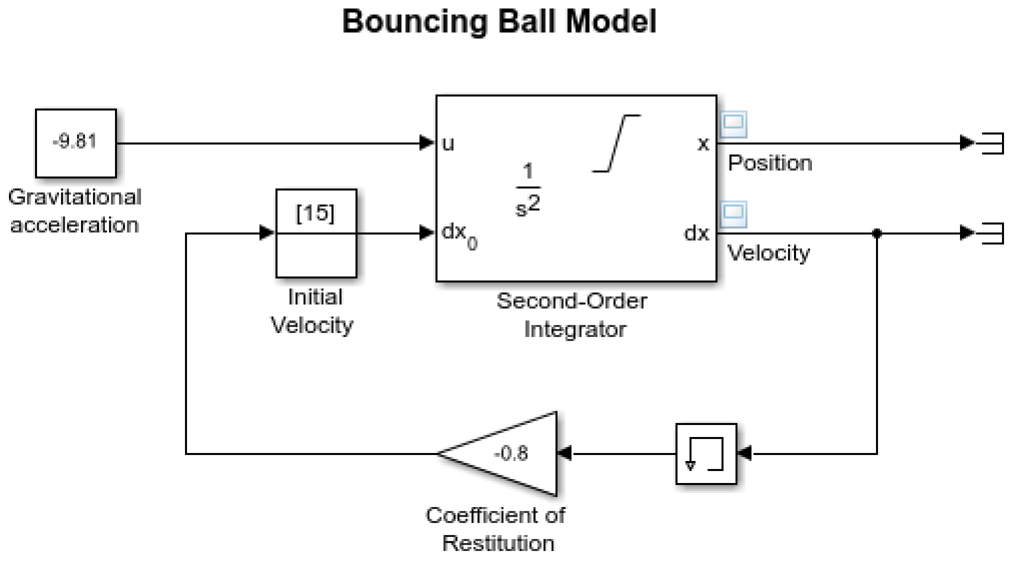


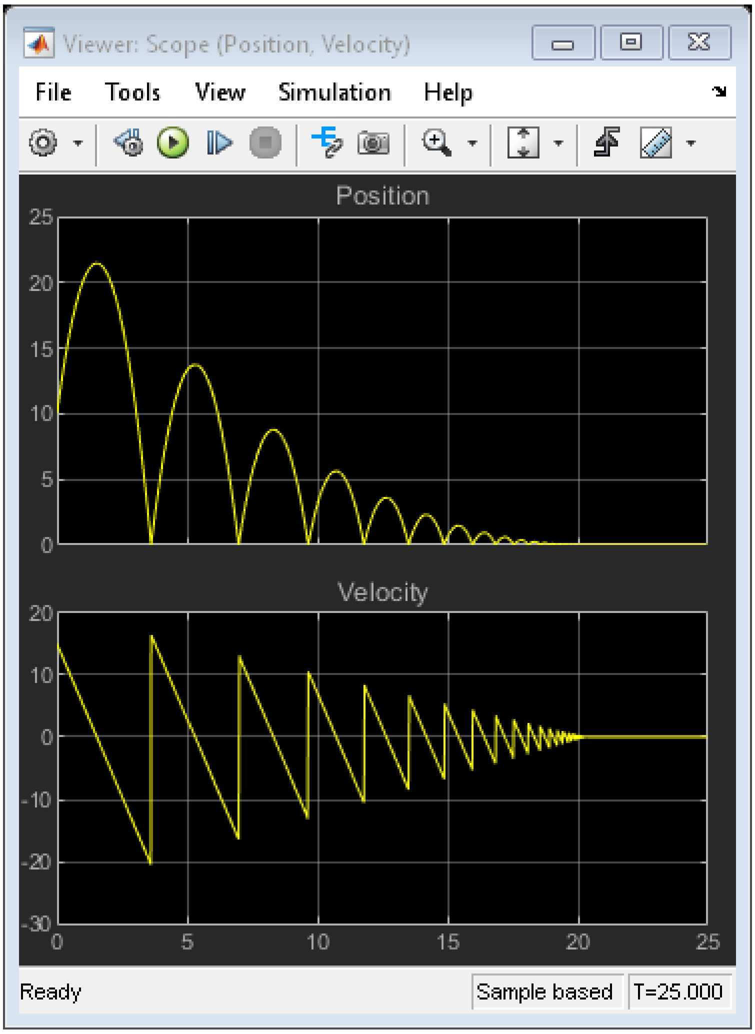
To observe the Zeno behavior of the system, navigate to the Solver pane of the Configuration Parameters dialog box. In the 'Zero-crossing options' section, confirm that 'Algorithm' is set to 'Nonadaptive' and that the simulation 'Stop time' is set to 25 seconds. Run the simulation.

Observe that the simulation errors out as the ball hits the ground more and more frequently and loses energy. Consequently, the simulation exceeds the default limit of 1000 for the 'Number of consecutive zero crossings' allowed. Now navigate to the Configuration Parameters dialog box. In the 'Zero-crossing options' section, set the 'Algorithm' to 'Adaptive'. This algorithm introduces a sophisticated treatment of such chattering behavior. Therefore, you can now simulate the system beyond 20 seconds. Note, however, the chatter of the states between 21 seconds and 25 seconds and warning from Simulink about the strong chattering in the model around 20 seconds.

Using a Second-Order Integrator Block to Model a Bouncing Ball

You can use a single Second-Order Integrator block to model this system. The second equation $dx/dt = v$ is internal to the Second-Order Integrator block in this case. Navigate to the Second-Order Integrator block dialog and notice that, as earlier, x has a lower limit of zero. Navigate to the Attributes tab on the block dialog and note that the option 'Reinitialize dx/dt when x reaches saturation' is checked. This parameter allows us to reinitialize dx/dt (v in the bouncing ball model) to a new value at the instant x reaches its saturation limit. For the bouncing ball model, this option therefore implies that when the ball hits the ground, its velocity can be set to a different value, i.e., to the velocity after the impact. Notice the loop for calculating the velocity after a collision with the ground. To capture the velocity v^- of the ball just before the collision, the dx/dt output port of the Second-Order Integrator block and a Memory block are used. v^- is then used to calculate the rebound velocity v^+.





Navigate to the Solver pane of the Configuration Parameters dialog box. Confirm that 'Algorithm' is set to 'Nonadaptive' in the 'Zero-crossing options' section and the simulation 'Stop Time' is set to 25 seconds. Simulate the model. Note that the simulation encountered no problems. You were able to simulate the model without experiencing excessive chatter after t = 20 seconds and without setting the 'Algorithm' to 'Adaptive'.

Second-Order Integrator Model Is the Preferable Approach to Modeling Bouncing Ball

One can analytically calculate the exact time t^* when the ball settles down to the ground with zero velocity by summing the time required for each bounce. This time is the sum of an infinite geometric series given by:

$$t^* = \frac{1}{g}\left(v_0 + v_1\left(\frac{1+\kappa}{1-\kappa}\right)\right), \qquad v_1 = \sqrt{v_0^2 + 2gx_0}.$$

Here x_0 and v_0 are initial conditions for position and velocity respectively. The velocity and the position of the ball must be identically zero for $t > t^*$. In the figure below, results from both simulations are plotted near t^*. The vertical red line in the plot is t^* for the given model parameters. For $t < t^*$ and far away from t^*, both models produce accurate and identical results. Hence, only a magenta line from the second model is visible in the plot. However, the simulation results from the first model are inexact after t^*; it continues to display excessive chattering behavior for $t > t^*$. In contrast, the second model using the Second-Order Integrator block settles to exactly zero for $t > t^*$.

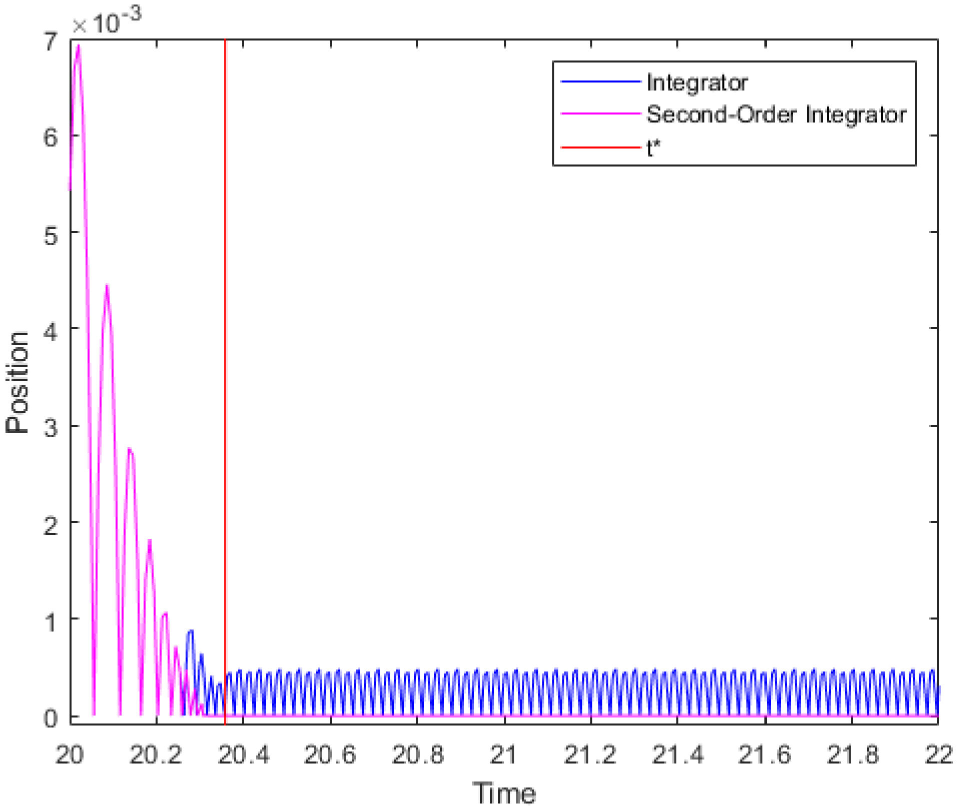


Figure 2: Comparison of simulation results from the two approaches.

Figure 2 conclusively shows that the second model has superior numerical characteristics as compared to the first model. The reason for the higher accuracy associated with the Second-Order Integrator model is as follows. The second differential equation $dx/dt = v$ is internal to the Second-Order Integrator block. Therefore, the block algorithms can leverage this known relationship between the two states and deploy heuristics to clamp down the undesirable chattering behavior for certain conditions. These heuristics become active when the two states are no longer mutually consistent with each other due to integration errors and chattering behavior. You can thus use physical knowledge of the system to alleviate the problem of simulation getting stuck in a Zeno state for certain classes of Zeno models.

Simulating Systems with Variable Transport Delay Phenomena

This example shows two cases where you can use Simulink® to model variable transport delay phenomena.

Vertical Wheel Displacement on a One-Dimensional Car

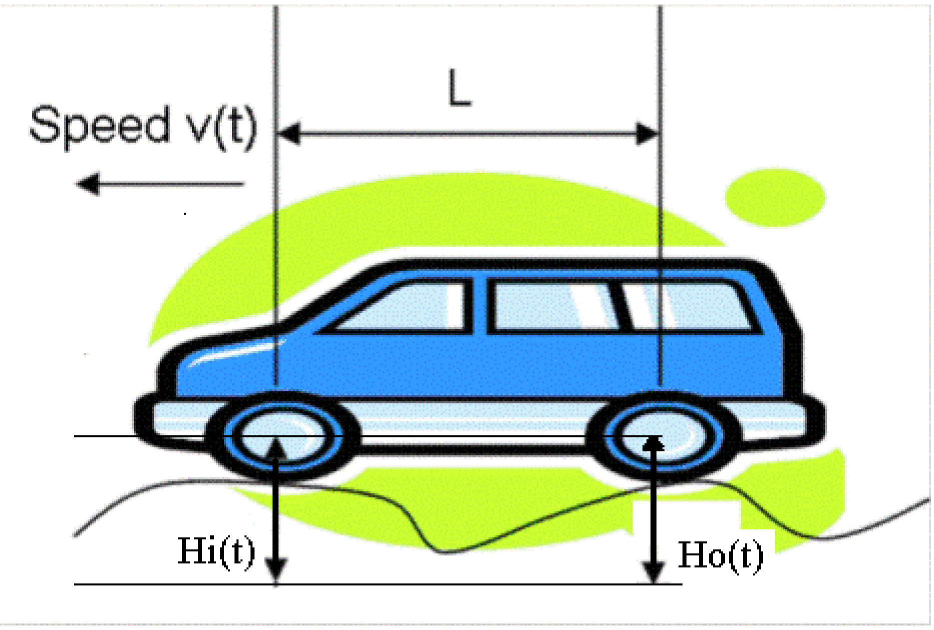
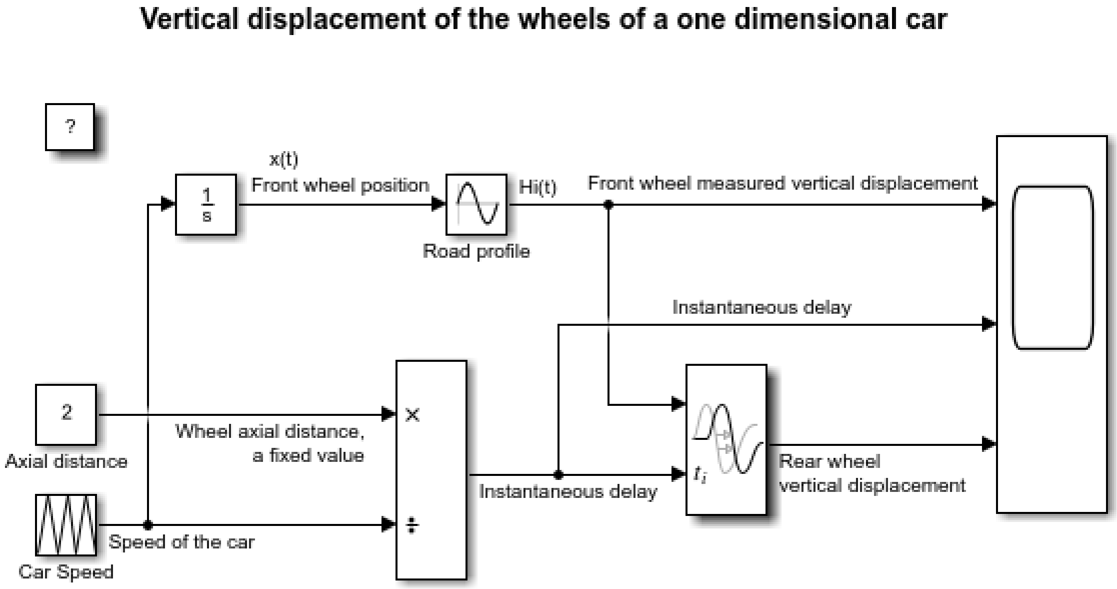


Figure 1: Illustration of a car with speed v(t).

A car is running along a road with speed v(t). A sensor is installed at the front wheel to measure the vertical displacement Hi(t) of the front wheel caused by the road profile. If the wheels and road never lose contact, then the vertical displacement of the rear wheel, Ho(t), can be seen as a variable transport delay of Hi(t), which is determined by the length L between the two wheels and the speed v(t).



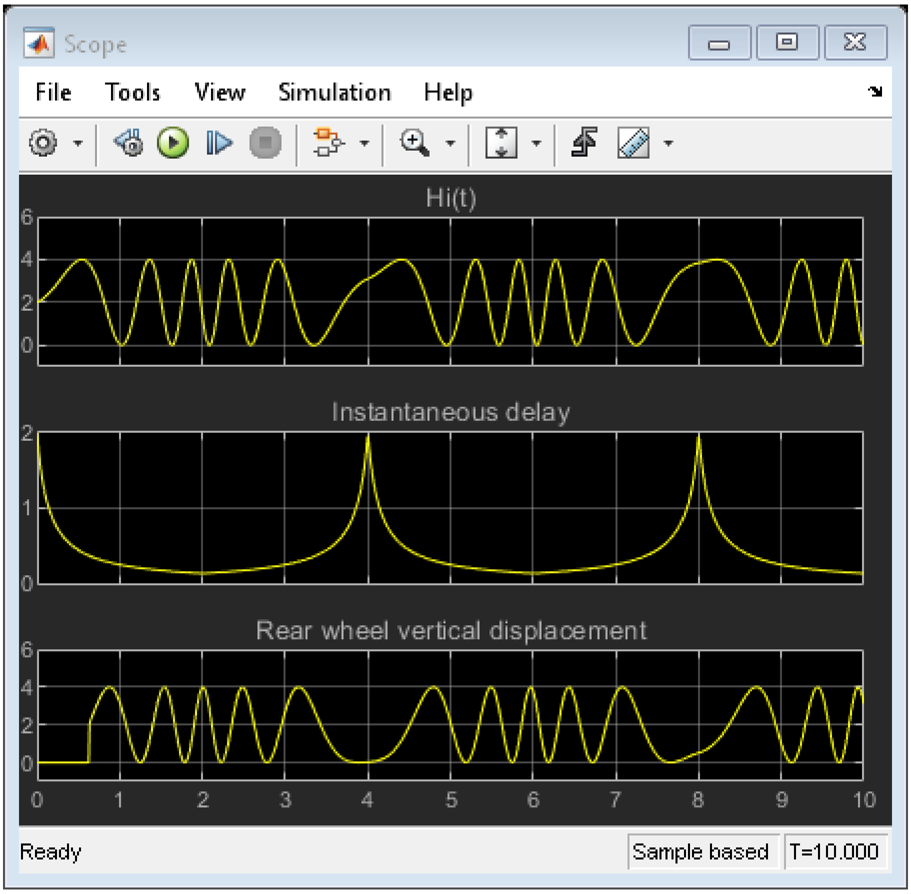


Figure 2: Vertical displacement of the wheels.

Incompressible Flow Through a Fixed Length Pipe

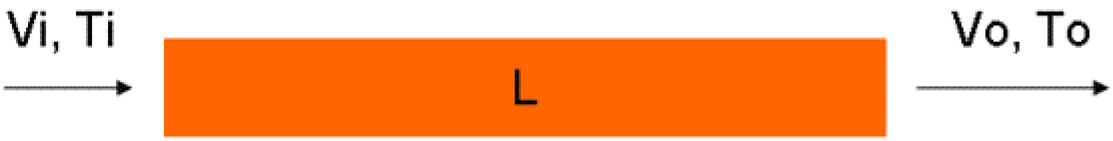
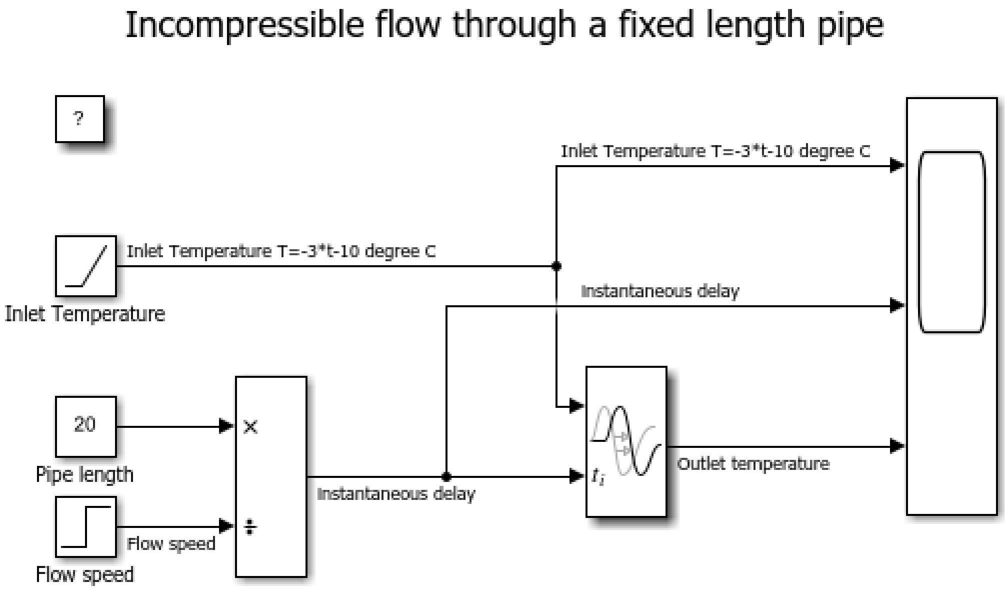


Figure 3: Illustration of a fixed-length pipe.

An incompressible flow goes through a pipe of length L with speed v(t). At the inlet, the flow temperature is Ti. We can model the temperature at the outlet To as a variable transport delay of Ti. At time t=0, the pipe is empty and until t=2, there is no flow at the outlet. Thus, the output temperature before t=2 is the initial output temperature.


Incompressible flow through a fixed length pipe
?
Inlet Temperature T=-3*t-10 degree C
Inlet Temperature T=-3*t-10 degree C
Inlet Temperature
Instantaneous delay
20
Pipe length
×
÷
Flow speed
Flow speed
Instantaneous delay
t_i
Outlet temperature

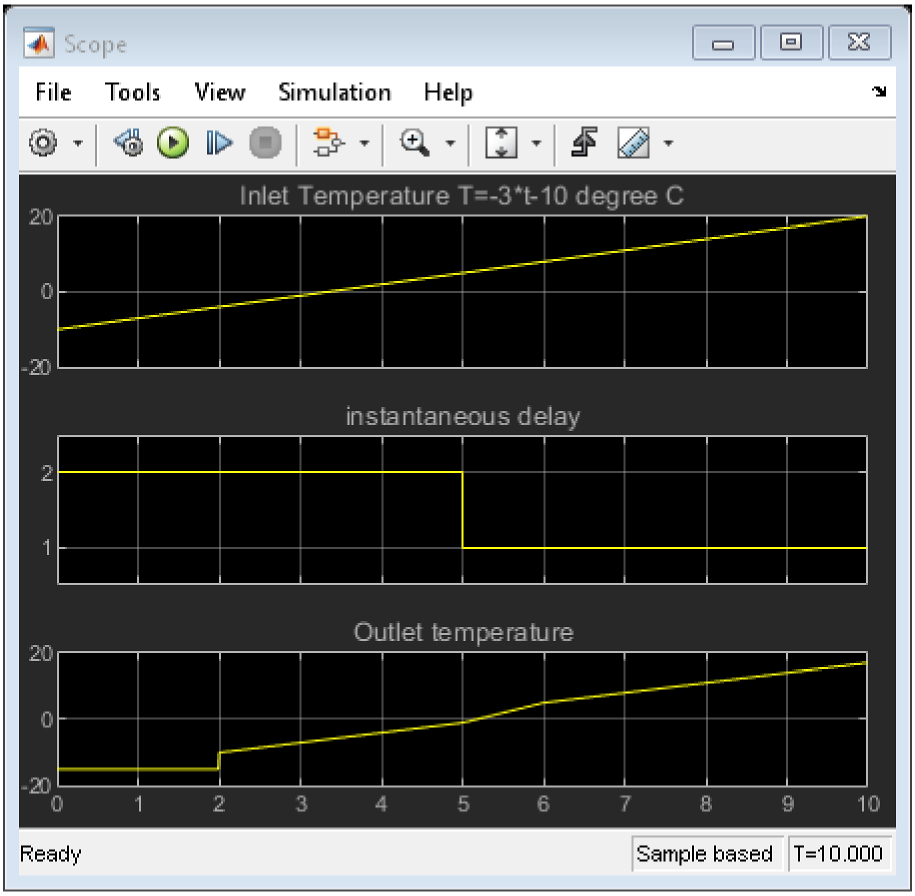


Figure 4: Incompressible flow through a fixed-length pipe.

Single Hydraulic Cylinder Simulation

This example shows how to use Simulink® to model a hydraulic cylinder. You can apply these concepts to applications where you need to model hydraulic behavior.

Analysis and Physics of the Model

Figure 1 shows a schematic diagram of the basic model. The model directs the pump flow, `Q`, to supply pressure, `p1`, from which laminar flow, `q1ex`, leaks to exhaust. The control valve for the piston/cylinder assembly is modeled as turbulent flow through a variable-area orifice. Its flow, `q12`, leads to intermediate pressure, `p2`, which undergoes a subsequent pressure drop in the line connecting it to the actuator cylinder. The cylinder pressure, `p3`, moves the piston against a spring load, resulting in position `x`.

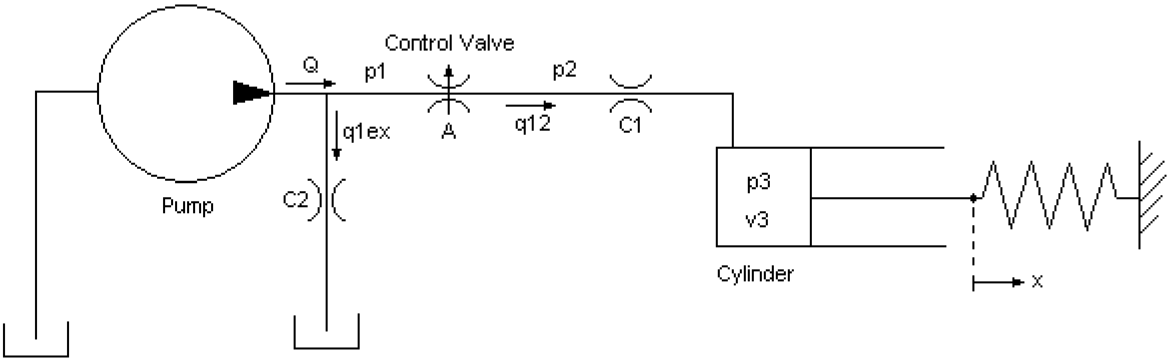


Figure 1: Schematic diagram of the basic hydraulic system

At the pump output, the flow is split between leakage and flow to the control valve. We model the leakage, `q1ex`, as laminar flow (see Equation Block 1).

Equation Block 1

$$Q = q_{12} + q_{1ex}$$

$$q_{1ex} = C_2 \cdot p_1$$

$$p_1 = \frac{(Q - q_{12})}{C_2}$$

$$Q = \text{ pump flow}$$

$$q_{12} = \text{ control valve flow}$$

$$q_{1ex} = \text{ leakage}$$

$$C_2 = \text{ flow coefficient}$$

$$p_1 = \text{ pump pressure}$$

We modeled turbulent flow through the control valve with the orifice equation. The sign and absolute value functions accommodate flow in either direction (see Equation Block 2).

Equation Block 2

$$q_{12} = C_d \cdot A \cdot sgn(p_1 - p_2) \cdot \sqrt{\frac{2}{\rho}|p_1 - p_2|}$$

$$C_d = \text{ orifice discharge coefficient}$$

$$A = \text{ orifice area}$$

$$p_2 = \text{ pressure downstream of control valve}$$

$$\rho = \text{ fluid density}$$

The fluid within the cylinder pressurizes due to this flow, `q12 = q23`, minus the compliance of the piston motion. We also modeled fluid compressibility in this case (see Equation Block 3).

Equation Block 3

$$\frac{dp_3}{dt} = \frac{\beta}{V_3}\left(q_{12} - A_c\frac{dx}{dt}\right)$$

$$V_3 = V_{30} + A_c \cdot x$$

$$p_3 = \text{ piston pressure}$$

$$\beta = \text{ fluid bulk modulus}$$

$$V_3 = \text{ fluid volume at } p_3$$

$$V_{30} = \text{ fluid volume in the piston for } x = 0$$

$$A_c = \text{ cylinder cross-sectional area}$$

We neglected the piston and spring masses because of the large hydraulic forces. We completed the system of equations by differentiating this relationship and incorporating the pressure drop between `p2` and `p3`. Equation Block 3 models laminar flow in the line from the valve to the actuator. Equation block 4 gives the force balance at the piston.

Equation Block 4

$$x = p_3\frac{A_c}{K}$$

$$\frac{dx}{dt} = \frac{dp_3}{dt}\frac{A_c}{K}$$

$$q_{23} = q_{12} = C_1\left(p_2 - p_3\right)$$

$$p_2 = p_3 + \frac{q_{12}}{C_1}$$

$$K = \text{ spring constant}$$

$$C_1 = \text{ laminar flow coefficient}$$

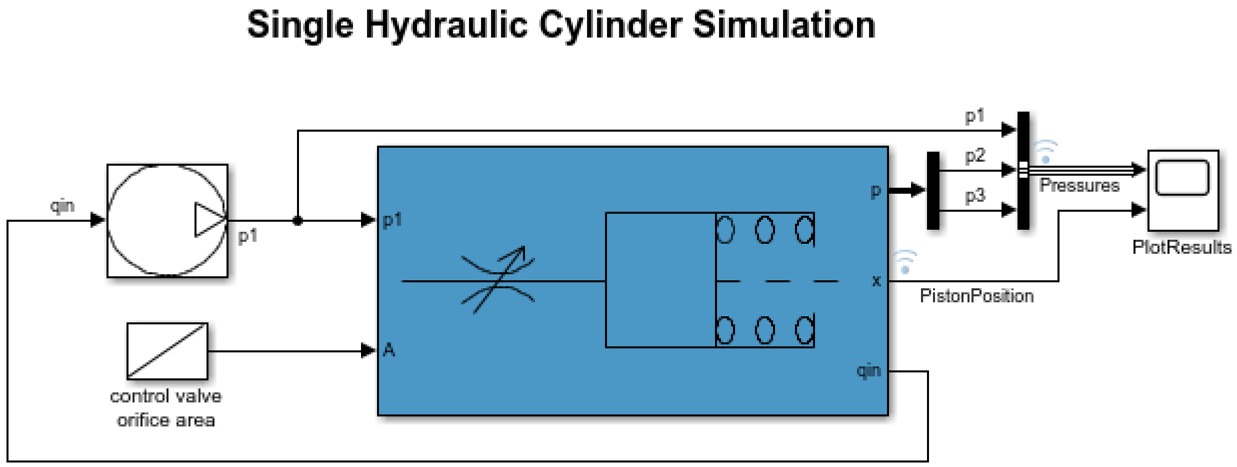
Modeling

Figure 2 shows the top level diagram of the model. The pump flow and the control valve orifice area are simulation inputs. The model is organized as two subsystems: the 'Pump' and the 'Valve/Cylinder/Piston/Spring Assembly'.

Opening the Model and Running the Simulation

To try it in MATLAB, type `sldemo_hydcyl` at MATLAB® terminal (click on the hyperlink if you are using MATLAB Help). Press the "Play" button on the model toolbar to run the simulation.

- Note: The model logs relevant data to MATLAB workspace in a structure called `sldemo_hydcyl_output`. Logged signals have a blue indicator.



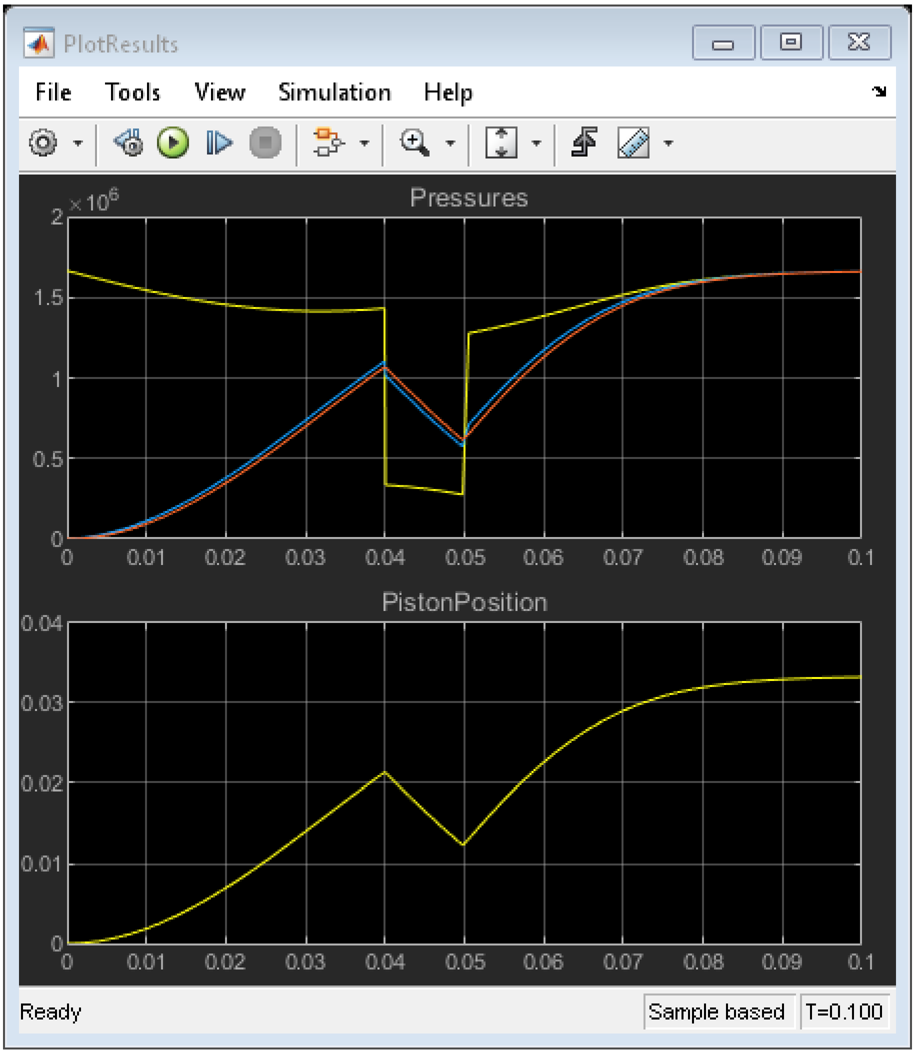


Figure 2: Single cylinder model and simulation results

'Pump' Subsystem

Right click on the 'Pump' masked subsystem and select "Look Under Mask" to see its components. The pump model computes the supply pressure as a function of the pump flow and the load (output) flow (Figure 3). `Qpump` is the pump flow data (saved in the model workspace). A matrix with column vectors of time points and the corresponding flow rates `[T, Q]`specifies the flow data. The model calculates pressure `p1` as indicated in Equation Block 1. Because `Qout = q12` is a direct function of `p1` (via the control valve), an algebraic loop is formed. An estimate of the initial value, `p10`, enables a more efficient solution.

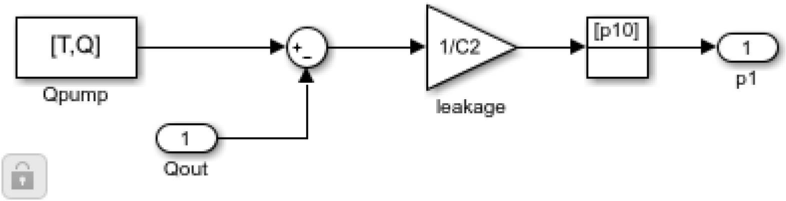
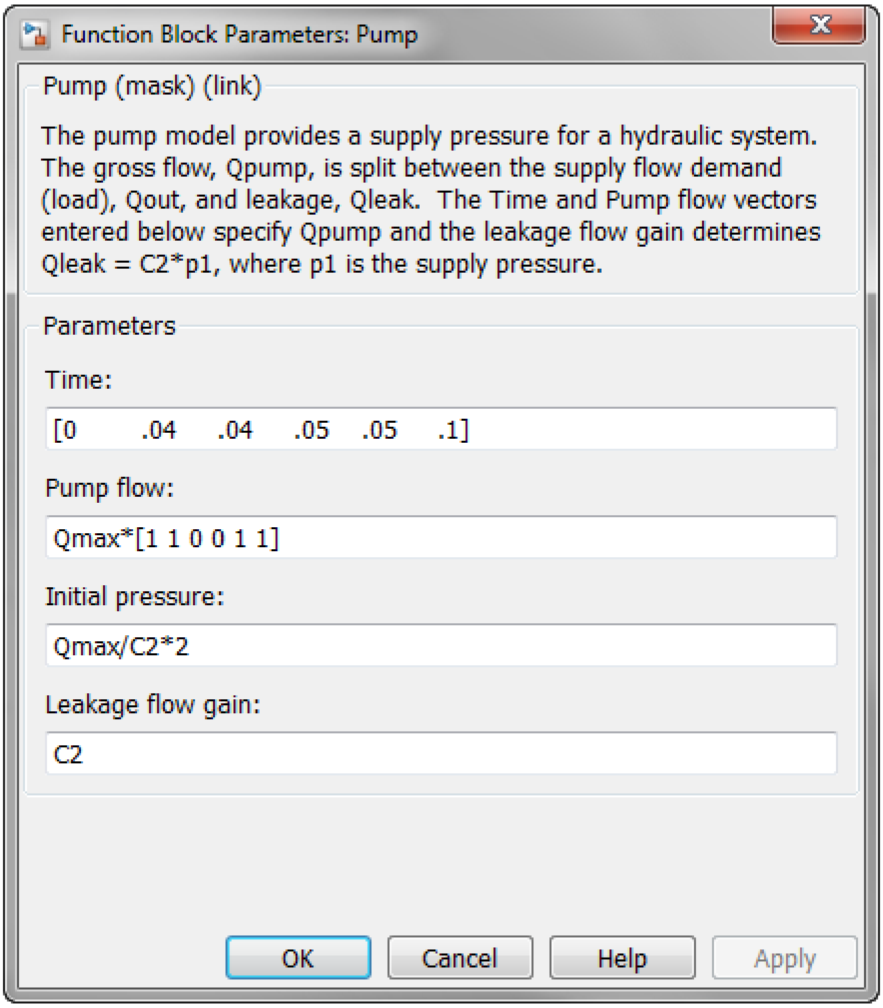


Figure 3: The pump subsystem

We masked the 'Pump' subsystem in Simulink to allow the user to easily access the parameters (see Figure 4). The parameters to be specified are `T`, `Q`, `p10`, and `C2`. We then assigned the masked block the icon shown in Figure 2, and saved it in a Simulink library.



Pump (mask) (link)

The pump model provides a supply pressure for a hydraulic system. The gross flow, Qpump, is split between the supply flow demand (load), Qout, and leakage, Qleak. The Time and Pump flow vectors entered below specify Qpump and the leakage flow gain determines Qleak = C2*p1, where p1 is the supply pressure.

Parameters

Time:

[0 .04 .04 .05 .05 .1]

Pump flow:

Qmax*[1 1 0 0 1 1]

Initial pressure:

Qmax/C2*2

Leakage flow gain:

C2

Figure 4: Entering pump parameters

'Valve/Cylinder/Piston/Spring Assembly' Subsystem

Right click on the 'Valve/Cylinder/Piston/Spring Assembly' in the model and select "Look Under Mask" to see the Actuator subsystem (see Figure 5). A system of differential-algebraic equations models the cylinder pressurization with the pressure `p3`, which appears as a derivative in Equation Block 3 and is used as the state (integrator). If we neglect piston mass, the spring force and piston position are direct multiples of `p3` and the velocity is a direct multiple of `p3`'s time derivative. This latter relationship forms an algebraic loop around the 'Beta' Gain block. The intermediate pressure `p2` is the sum of `p3` and the pressure drop due to the flow from the valve to the cylinder (Equation Block 4). This relationship also imposes an algebraic constraint through the control valve and the `1/C1` gain.

The control valve subsystem computes the orifice (Equation Block2). It uses as inputs the upstream and downstream pressures and the variable orifice area. The 'Control Valve Flow' Subsystem computes the signed square root:

$$y = sgn(u)\sqrt{|u|}$$

Three nonlinear functions are used, two of which are discontinuous. In combination, however, y is a continuous function of u.

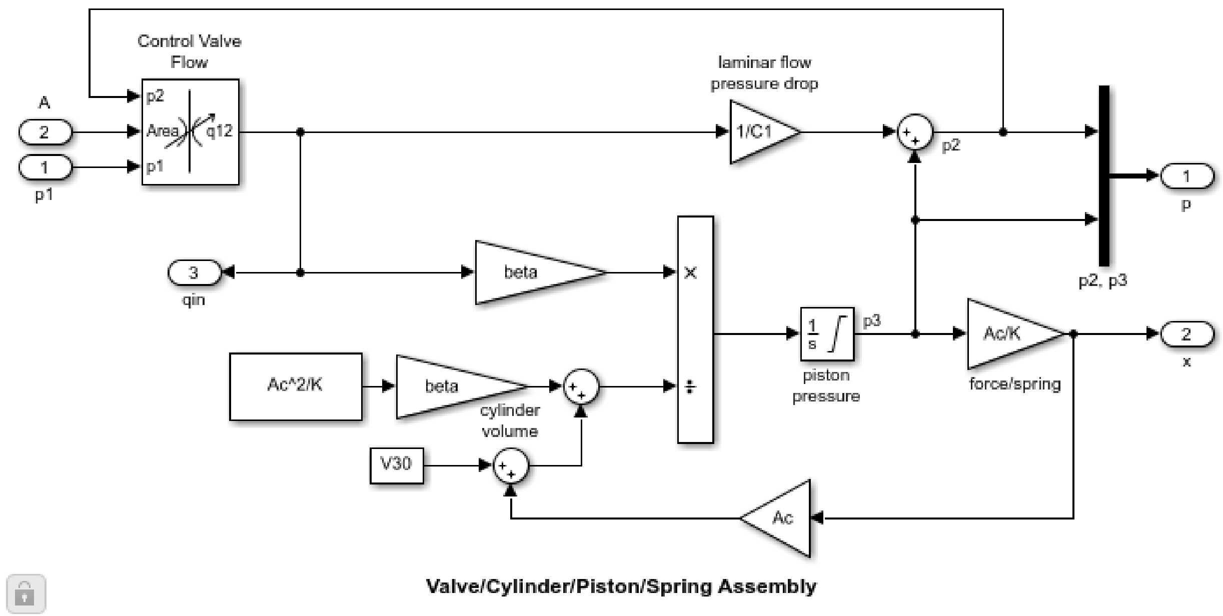


Figure 5: The valve/cylinder/piston/spring subsystem

Results

Simulation Parameters

We simulated the model using the following data. The information is loaded from a MAT-file - `sldemo_hydcyl_data.mat`, which is also used for the other two hydraulic cylinder models. The users can enter data via the Pump and Cylinder Masks shown in Figures 4 and 6.

$$C_d = 0.61$$

$$\rho = 800kg/m^3$$

$$C_1 = 2e-8m^3/sec/Pa$$

$$C_2 = 3e-9m^3/sec/Pa$$

$$\beta = 7e8Pa$$

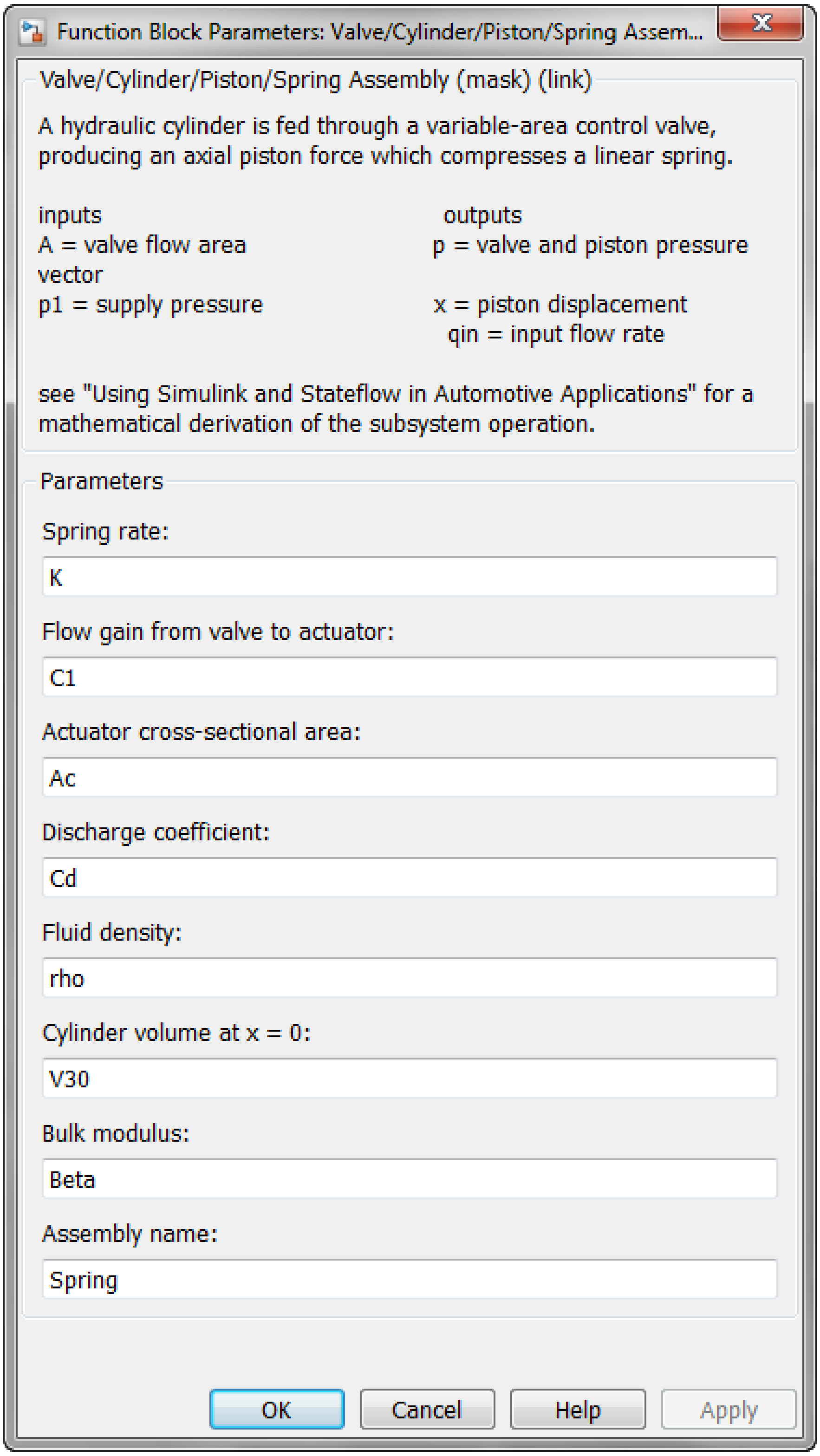
$$A_c = 1e-3m^2$$

$$K = 5e4N/m$$

$$V_{30} = 2.5e-5m^3$$

`T = [0 0.04 0.04 0.05 0.05 0.1 ] sec`

`Q = [0.005 0.005 0 0 0.005 0.005] m^3/sec`



Function Block Parameters: Valve/Cylinder/Piston/Spring Assem...

Valve/Cylinder/Piston/Spring Assembly (mask) (link)

A hydraulic cylinder is fed through a variable-area control valve, producing an axial piston force which compresses a linear spring.

```
inputs                                  outputs
A = valve flow area                    p = valve and piston pressure
vector
p1 = supply pressure                   x = piston displacement
                                         qin = input flow rate
```

see "Using Simulink and Stateflow in Automotive Applications" for a mathematical derivation of the subsystem operation.

Parameters

Spring rate: K

Flow gain from valve to actuator: C1

Actuator cross-sectional area: Ac

Discharge coefficient: Cd

Fluid density: rho

Cylinder volume at x = 0: V30

Bulk modulus: Beta

Assembly name: Spring

Figure 6: Entering valve/cylinder/piston/spring assembly parameters

Plotting Simulation Results

The system initially steps to a pump flow of `0.005 m^3/sec=300 l/min`, abruptly steps to zero at `t=0.04 sec`, then resumes its initial flow rate at `t=0.05 sec`.

The control valve starts with zero orifice area and ramps to `1e-4 sq.m.` during the `0.1 sec` simulation time. With the valve closed, all of the pump flow goes to leakage so the initial pump pressure increases to `p10 = Q/C2 = 1667 kPa`.

As the valve opens, pressures `p2` and `p3` build up while `p1` decreases in response to the load increase as shown in Figure 7. When the pump flow cuts off, the spring and piston act like an accumulator and `p3` decreases continuously. Then the flow reverses direction, so `p2`, though relatively close to `p3`, falls abruptly. At the pump itself, all of the back-flow leaks and `p1` drops radically. The behavior reverses as the flow is restored.

The piston position is directly proportional to `p3`, where the hydraulic and spring forces balance. Discontinuities in the velocity at `0.04` sec and `0.05` sec indicate negligible mass. The model reaches a steady state when all of the pump flow again goes to leakage, now due to zero pressure drop across the control valve (which means `p3 = p2 = p1 = p10`).

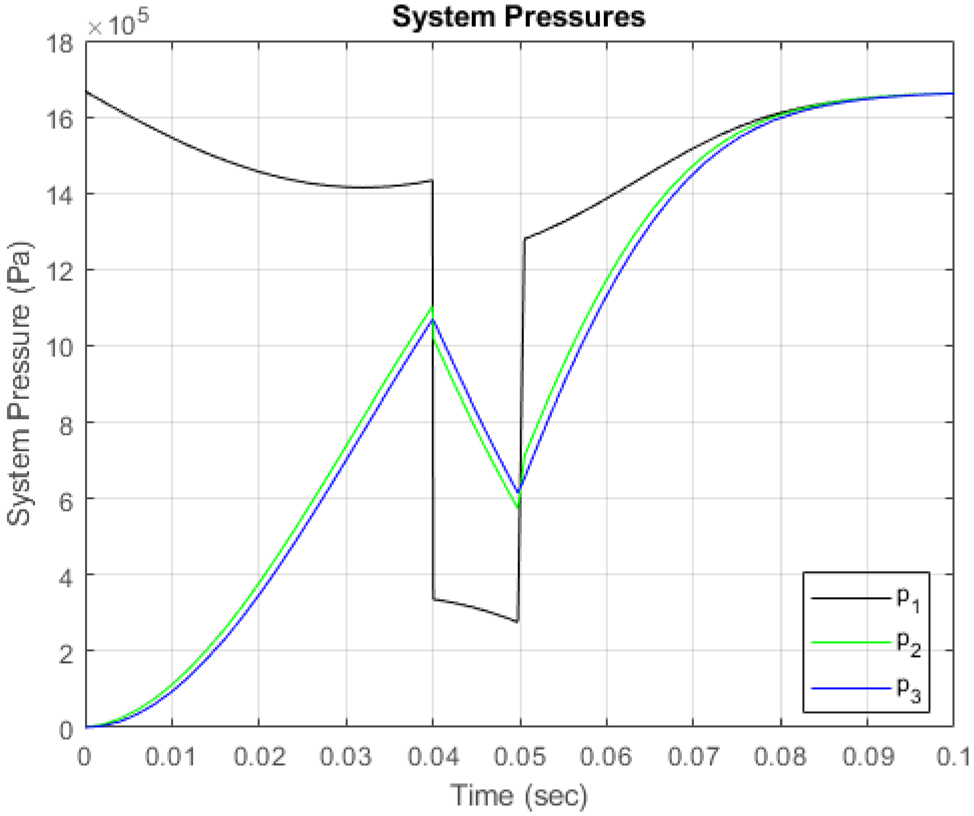


Figure 7: Simulation Results: System Pressures

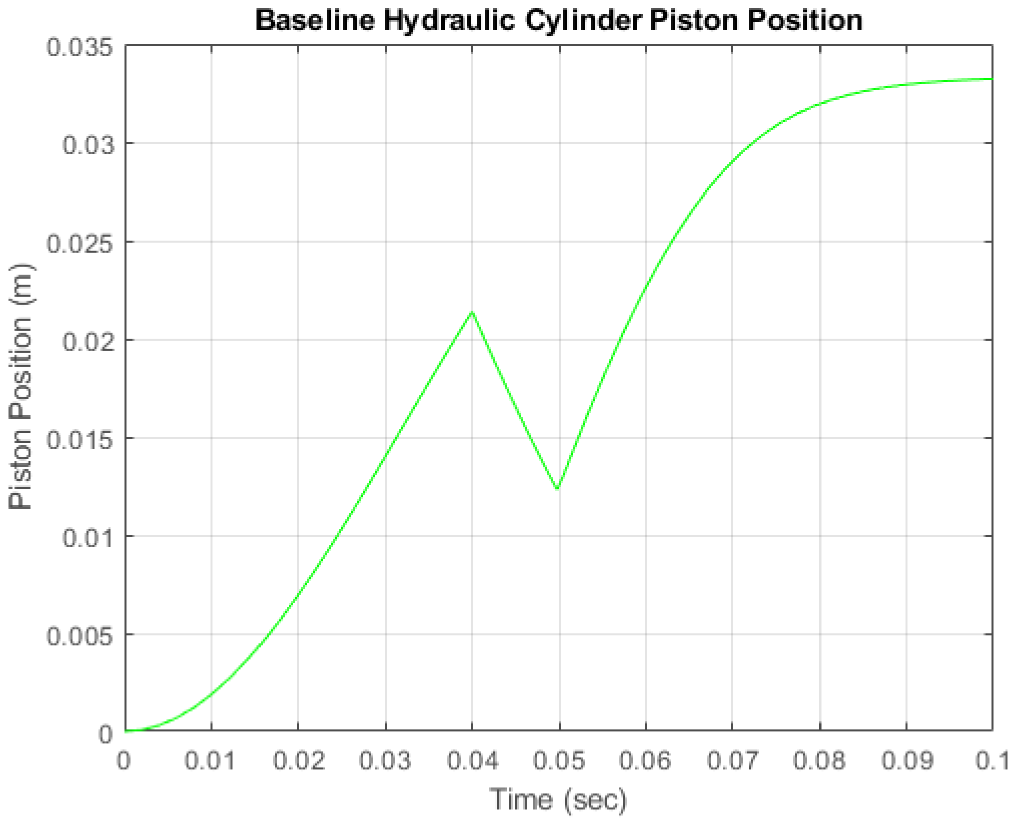


Figure 8: Simulation Results: Hydraulic Cylinder Piston Position

Closing the Model

Close the model and clear generated data.

Thermal Model of a House

This example shows how to use Simulink® to create the thermal model of a house. This system models the outdoor environment, the thermal characteristics of the house, and the house heating system.

The sldemo_househeat_data.m file initializes data in the model workspace. To make changes, you can edit the model workspace directly or edit the file and re-load the model workspace. To view the model workspace, select View > Model Explorer from the Simulink editor.

Opening the Model

Open the sldemo_househeat model

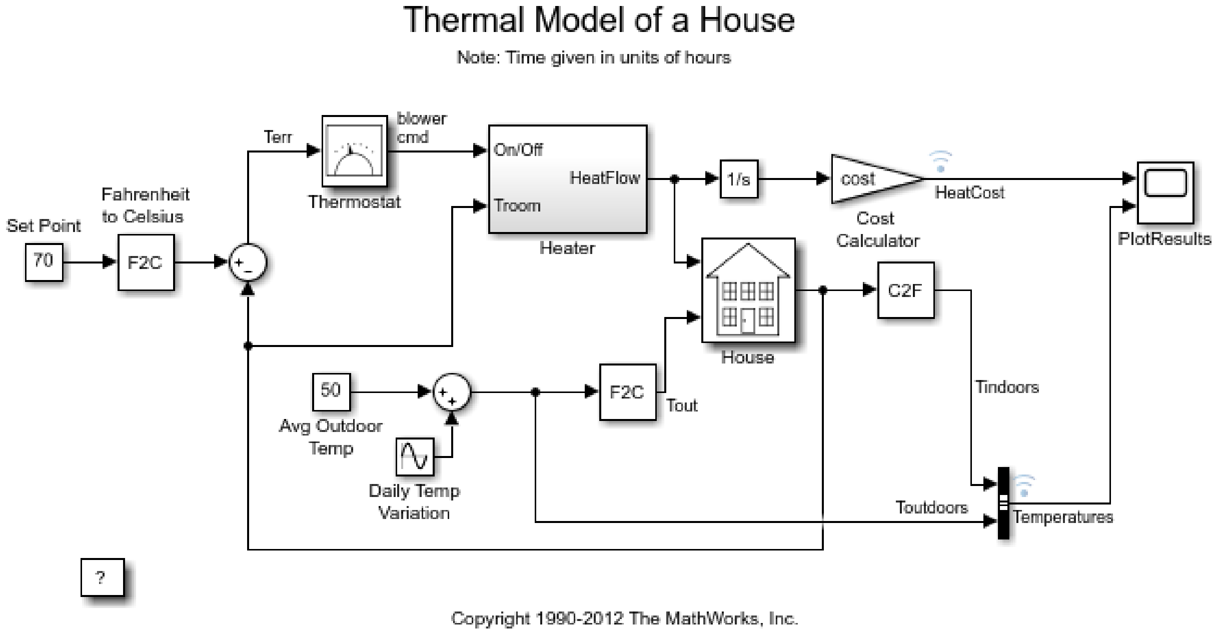


Figure 1: The House Heating Model

Model Initialization

This model calculates heating costs for a generic house. When the model is opened, it loads the information about the house from the sldemo_househeat_data.m file. The file does the following:

- Defines the house geometry (size, number of windows)
- Specifies the thermal properties of house materials
- Calculates the thermal resistance of the house
- Provides the heater characteristics (temperature of the hot air, flow-rate)
- Defines the cost of electricity (0.09$/kWhr)
- Specifies the initial room temperature (20 deg. Celsius = 68 deg. Fahrenheit)
- **Note:** Time is given in units of hours. Certain quantities, like air flow-rate, are expressed per hour (not per second).

Model Components

Set Point

"Set Point" is a constant block. It specifies the temperature that must be maintained indoors. It is 70 degrees Fahrenheit by default. Temperatures are given in Fahrenheit, but then are converted to Celsius to perform the calculations.

Thermostat

"Thermostat" is a subsystem that contains a Relay block. The thermostat allows fluctuations of 5 degrees Fahrenheit above or below the desired room temperature. If air temperature drops below 65 degrees Fahrenheit, the thermostat turns on the heater. See the thermostat subsystem below.

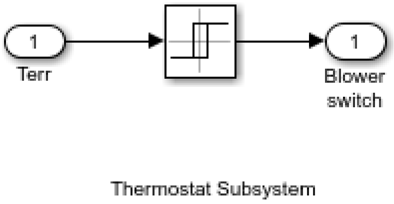


Figure 2: The "Thermostat" Subsystem

Heater

"Heater" is a subsystem that has a constant air flow rate, "Mdot", which is specified in the sldemo_househeat_data.m file. The thermostat signal turns the heater on or off. When the heater is on, it blows hot air at temperature THeater (50 degrees Celsius = 122 degrees Fahrenheit by default) at a constant flow rate of Mdot (1kg/sec = 3600kg/hr by default). The heat flow into the room is expressed by the Equation 1.

Equation 1

$$\frac{dQ}{dt} = (T_{heater} - T_{room}) \cdot Mdot \cdot c$$

$$\frac{dQ}{dt} = \text{ heat flow from the heater into the room}$$

$$c = \text{ heat capacity of air at constant pressure}$$

$$Mdot = \text{ air mass flow rate through heater (kg/hr)}$$

$$T_{heater} = \text{ temperature of hot air from heater}$$

$$T_{room} = \text{ current room air temperature}$$

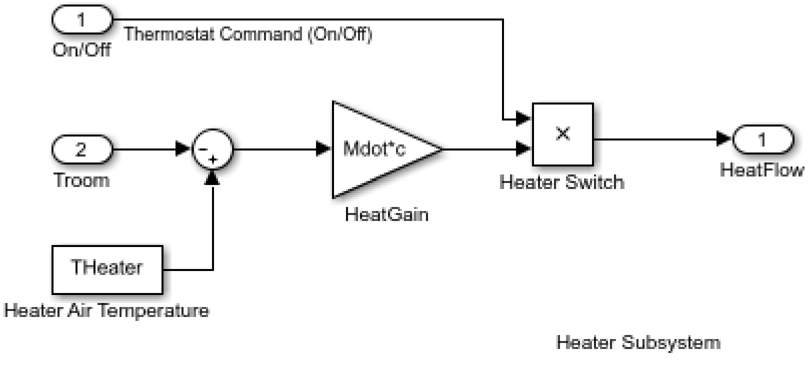


Figure 3: The Heater Subsystem

Cost Calculator

"Cost Calculator" is a Gain block. "Cost Calculator" integrates the heat flow over time and multiplies it by the energy cost. The cost of heating is plotted in the "PlotResults" scope.

House

"House" is a subsystem that calculates room temperature variations. It takes into consideration the heat flow from the heater and heat losses to the environment. Heat losses and the temperature time derivative are expressed by Equation 2.

Equation 2

$$\left(\frac{dQ}{dt}\right)_{losses} = \frac{T_{room} - T_{out}}{R_{eq}}$$

$$\frac{dT_{room}}{dt} = \frac{1}{M_{air} \cdot c} \cdot \left(\frac{dQ_{heater}}{dt} - \frac{dQ_{losses}}{dt}\right)$$

$$M_{air} = \text{ mass of air inside the house}$$

$$R_{eq} = \text{ equivalent thermal resistance of the house}$$

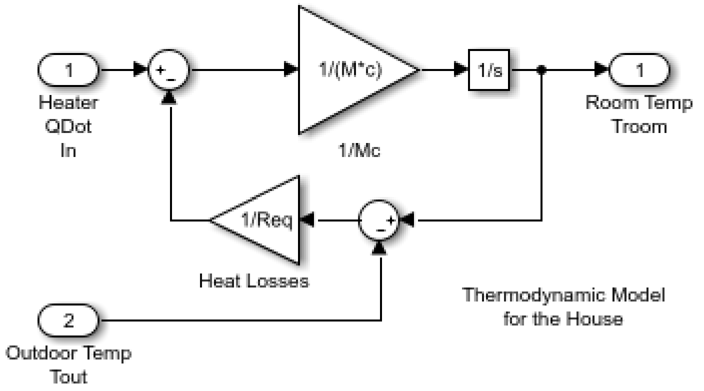


Figure 4: The House Subsystem

Modeling the Environment

We model the environment as a heat sink with infinite heat capacity and time varying temperature Tout. The constant block "Avg Outdoor Temp" specifies the average air temperature outdoors. The "Daily Temp Variation" Sine Wave block generates daily temperature fluctuations of outdoor temperature. Vary these parameters and see how they affect the heating costs.

Running the Simulation and Visualizing the Results

Run the simulation and visualize the results. Open the "PlotResults" scope to visualize the results. The heat cost and indoor versus outdoor temperatures are plotted on the scope. The temperature outdoor varies sinusoidally, whereas the indoors temperature is maintained within 5 degrees Fahrenheit of "Set Point". Time axis is labeled in hours.

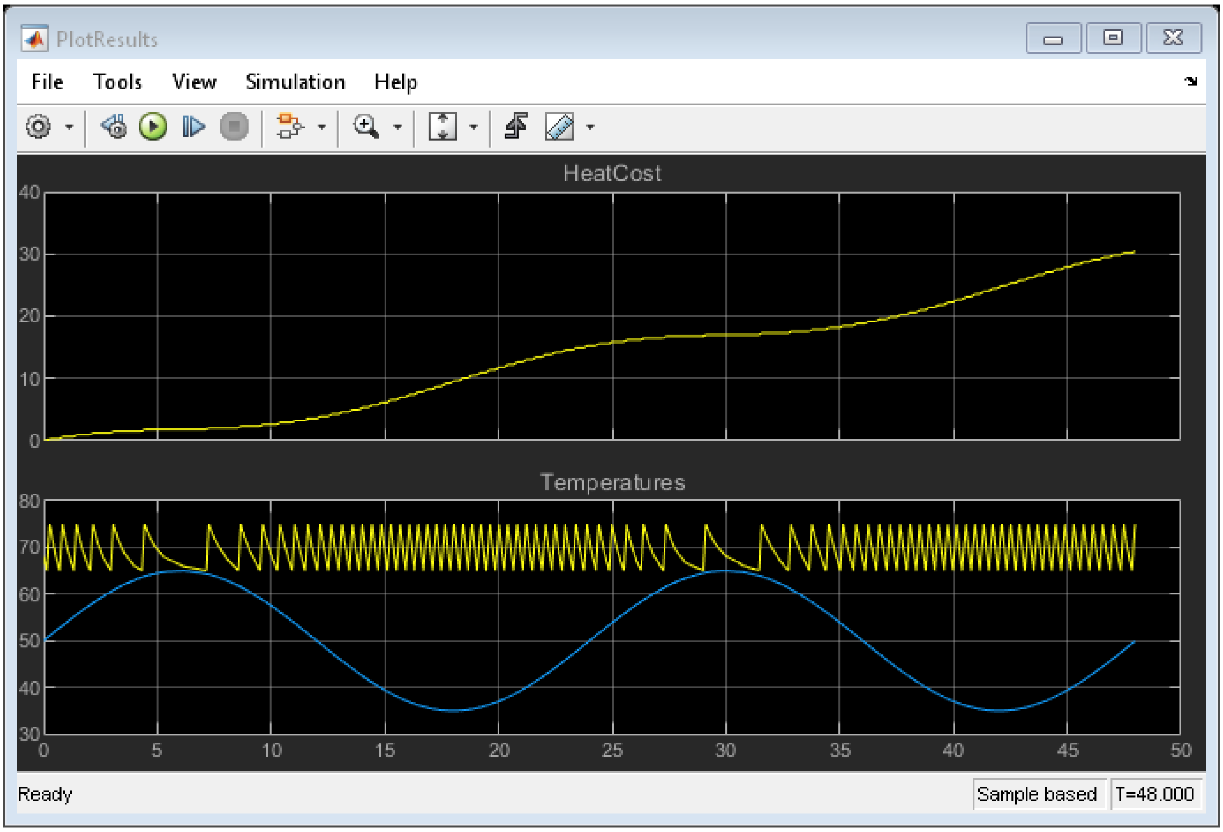


Figure 5: Simulation results (time axis labeled in hours)

According to this model, it would cost around $30 to heat the house for two days. Try varying the parameters and observe the system response.

Remarks

This particular model is designed to calculate the heating costs only. If the temperature of the outside air is higher than the room temperature, the room temperature will exceed the desired "Set Point".

You can modify this model to include an air conditioner. You can implement the air conditioner as a modified heater. To do this, add parameters like the following to sldemo_househeat_data.m.

- Cold air output
- Temperature of the stream from the air conditioner
- Air conditioner efficiency

You would also need to modify the thermostat to control both the air conditioner and the heater.

Modeling an Anti-Lock Braking System

This example shows how to model a simple model for an Anti-Lock Braking System (ABS). It simulates the dynamic behavior of a vehicle under hard braking conditions. The model represents a single wheel, which may be replicated a number of times to create a model for a multi-wheel vehicle.

This model uses the signal logging feature in Simulink®. The model logs signals to the MATLAB® workspace where you can analyze and view them. You can view the code in `sldemo_absbrakeplots.m` to see how this is done.

In this model, the wheel speed is calculated in a separate model named `sldemo_wheelspeed_absbrake`. This component is then referenced using a 'Model' block. Note that both the top model and the referenced model use a variable step solver, so Simulink will track zero-crossings in the referenced model.

Analysis and Physics

The wheel rotates with an initial angular speed that corresponds to the vehicle speed before the brakes are applied. We used separate integrators to compute wheel angular speed and vehicle speed. We use two speeds to calculate slip, which is determined by Equation 1. Note that we introduce vehicle speed expressed as an angular velocity (see below).

$$\omega_v = \frac{V}{R} \text{ (equals the wheel angular speed if there is no slip)}$$

Equation 1

$$\omega_v = \frac{V_v}{R_r}$$

$$slip = 1 - \frac{\omega_w}{\omega_v}$$

$$\omega_v = \text{ vehicle speed divided by wheel radius}$$

$$V_v = \text{ vehicle linear velocity}$$

$$R_r = \text{ wheel radius}$$

$$\omega_w = \text{ wheel angular velocity}$$

From these expressions, we see that slip is zero when wheel speed and vehicle speed are equal, and slip equals one when the wheel is locked. A desirable slip value is `0.2`, which means that the number of wheel revolutions equals `0.8` times the number of revolutions under non-braking conditions with the same vehicle velocity. This maximizes the adhesion between the tire and road and minimizes the stopping distance with the available friction.

Modeling

The friction coefficient between the tire and the road surface, `mu`, is an empirical function of slip, known as the mu-slip curve. We created mu-slip curves by passing MATLAB variables into the block diagram using a Simulink lookup table. The model multiplies the friction coefficient, `mu`, by the weight on the wheel, `W`, to yield the frictional force, `Ff`, acting on the circumference of the tire. `Ff` is divided by the vehicle mass to produce the vehicle deceleration, which the model integrates to obtain vehicle velocity.

In this model, we used an ideal anti-lock braking controller, that uses 'bang-bang' control based upon the error between actual slip and desired slip. We set the desired slip to the value of slip at which the mu-slip curve reaches a peak value, this being the optimum value for minimum braking distance (see note below.).

- Note: In an actual vehicle, the slip cannot be measured directly, so this control algorithm is not practical. It is used in this example to illustrate the conceptual construction of such a simulation model. The real engineering value of a simulation like this is to show the potential of the control concept prior to addressing the specific issues of implementation.

Creating a Temporary Directory for the Example

During this example, Simulink generates files in the current working directory. If you do not want to generate files in this directory, change the working directory to a suitable directory:

```
    origdir = cd(tempdir);
```

Opening the Model

To open this model type `sldemo_absbrake` in MATLAB terminal (or click on the hyperlink if you are using MATLAB Help).

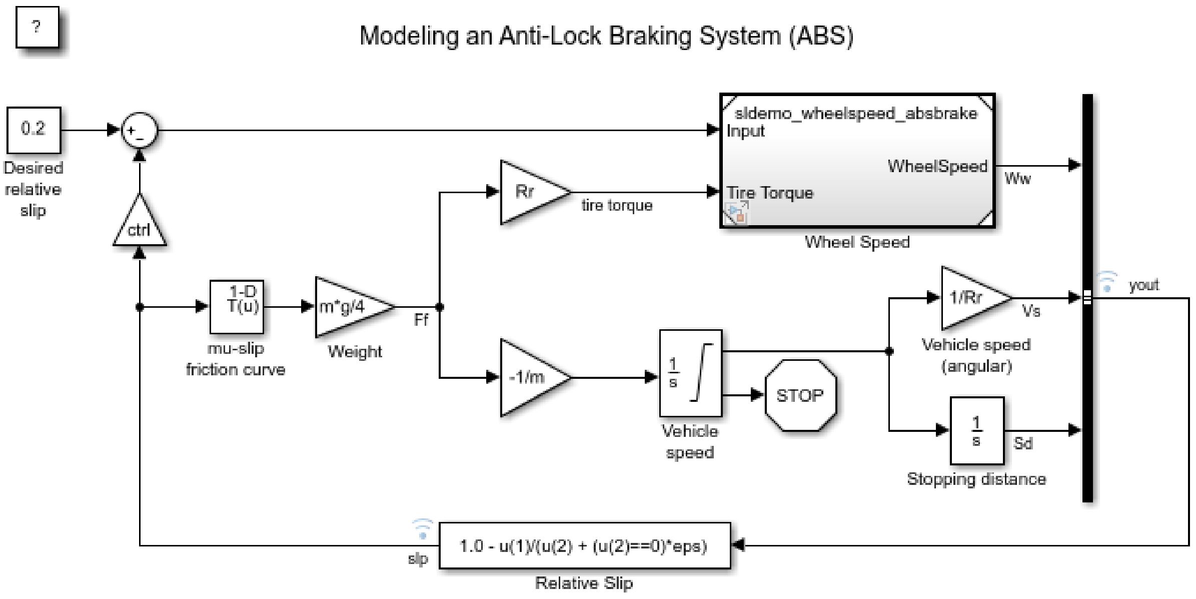


Figure 1: Anti-Lock Braking (ABS) Model

Double click on the 'Wheel Speed' subsystem in the model window to open it. Given the wheel slip, the desired wheel slip, and the tire torque, this subsystem calculates the wheel angular speed.

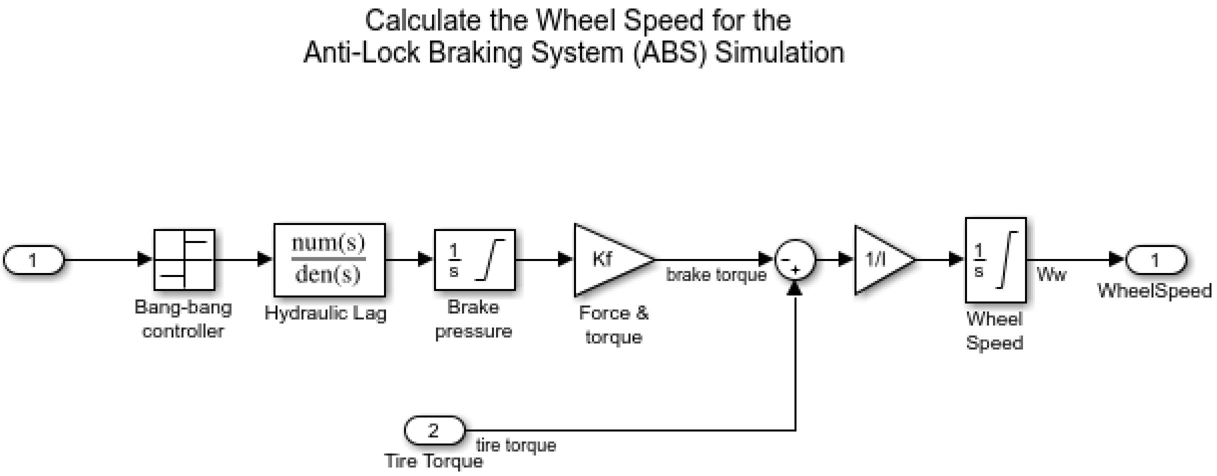


Figure 2: Wheel Speed subsystem

To control the rate of change of brake pressure, the model subtracts actual slip from the desired slip and feeds this signal into a bang-bang control (`+1` or `-1`, depending on the sign of the error, see Figure 2). This on/off rate passes through a first-order lag that represents the delay associated with the hydraulic lines of the brake system. The model then integrates the filtered rate to yield the actual brake pressure. The resulting signal, multiplied by the piston area and radius with respect to the wheel (`Kf`), is the brake torque applied to the wheel.

The model multiplies the frictional force on the wheel by the wheel radius (`Rr`) to give the accelerating torque of the road surface on the wheel. The brake torque is subtracted to give the net torque on the wheel. Dividing the net torque by the wheel rotational inertia, `I`, yields the wheel acceleration, which is then integrated to provide wheel velocity. In order to keep the wheel speed and vehicle speed positive, limited integrators are used in this model.

Running the Simulation in ABS Mode

Press the "Play" button on the model toolbar to run the simulation. You can also run the simulation by executing the `sim('sldemo_absbrake')` command in MATLAB. ABS is turned on during this simulation.

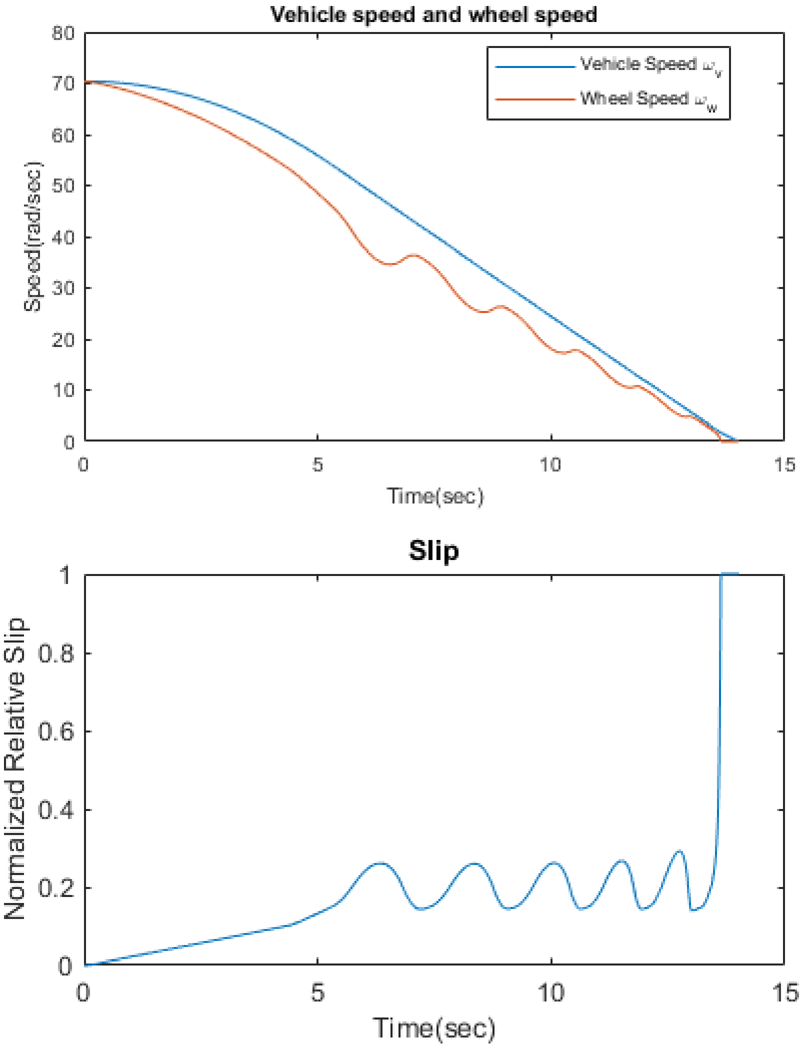


Figure 3: Baseline Simulation Results

- Note: The model logs relevant data to MATLAB workspace in a structure called `sldemo_absbrake_output`. Logged signals have a blue indicator. In this case `yout` and `slp` are logged (see the model). Read more about Signal Logging in Simulink Help.

Figure 3 visualizes the ABS simulation results (for default parameters). The first plot in Figure 3 shows the wheel angular velocity and corresponding vehicle angular velocity. This plot shows that the wheel speed stays below vehicle speed without locking up, with vehicle speed going to zero in less than 15 seconds.

Running the Simulation Without ABS

For more meaningful results, consider the vehicle behavior without ABS. At the MATLAB command line, set the model variable `ctrl = 0`. This disconnects the slip feedback from the controller (see Figure 1), resulting in maximum braking. The results are shown in Figure 4.

```
    ctrl = 0;
```

Now run the simulation again. This will model braking without ABS.

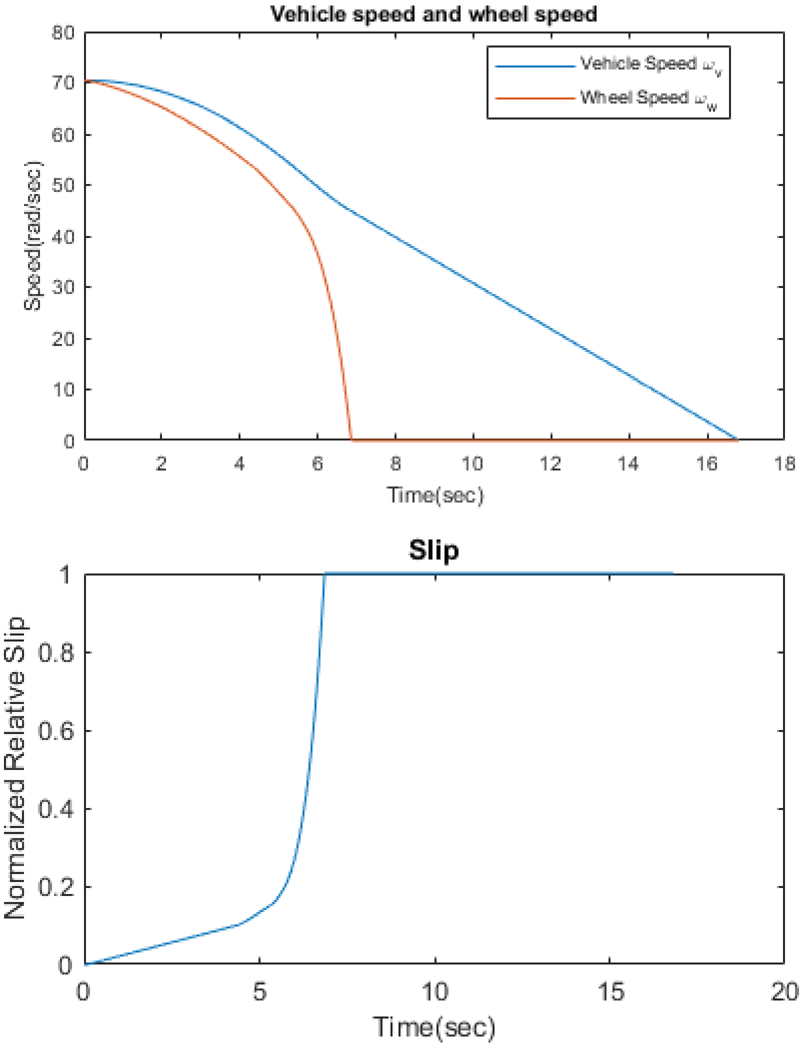


Figure 4: Maximum braking simulation results (braking without ABS)

Braking With ABS Versus Braking Without ABS

In the upper plot of Figure 4, observe that the wheel locks up in about seven seconds. The braking, from that point on, is applied in a less-than-optimal part of the slip curve. That is, when `slip = 1`, as seen in the lower plot of Figure 4, the tire is skidding so much on the pavement that the friction force has dropped off.

This is, perhaps, more meaningful in terms of the comparison shown in Figure 5. The distance traveled by the vehicle is plotted for the two cases. Without ABS, the vehicle skids about an extra 100 feet, taking about three seconds longer to come to a stop.

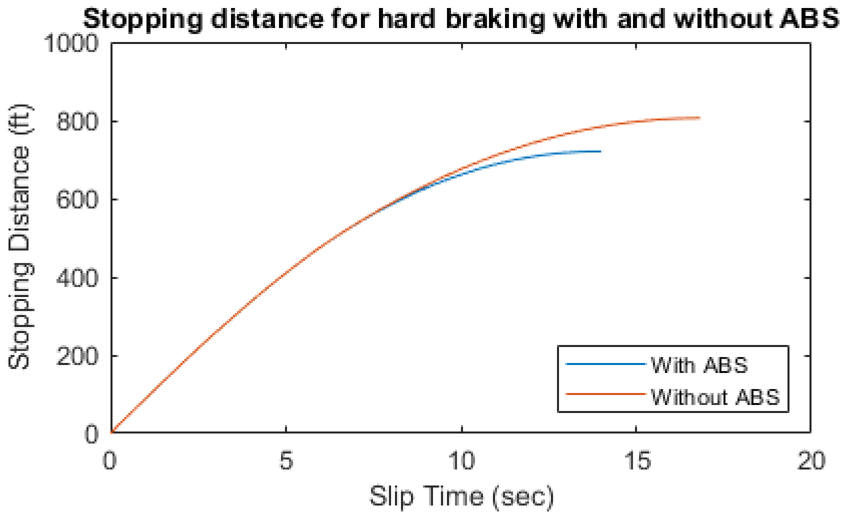


Figure 5: Stopping distance for hard braking with and without ABS

Closing the Model

Close the model. Close the 'Wheel Speed' subsystem. Clear logged data. Change back to the original directory.

```
cd(origdir);
```

Conclusions

This model shows how you can use Simulink to simulate a braking system under the action of an ABS controller. The controller in this example is idealized, but you can use any proposed control algorithm in its place to evaluate the system's performance. You can also use the Simulink® Coder™ with Simulink as a valuable tool for rapid prototyping of the proposed algorithm. C code is generated and compiled for the controller hardware to test the concept in a vehicle. This significantly reduces the time needed to prove new ideas by enabling actual testing early in the development cycle.

For a hardware-in-the-loop braking system simulation, you can remove the 'bang-bang' controller and run the equations of motion on real-time hardware to emulate the wheel and vehicle dynamics. You can do this by generating real-time C code for this model using the Simulink Coder. You can then test an actual ABS controller by interfacing it to the real-time hardware, which runs the generated code. In this scenario, the real-time model would send the wheel speed to the controller, and the controller would send brake action to the model.

BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXI No. 3/2009	93 - 98	Seria Tehnică

An Adjusted Mathematical Model for Realistic Road Traffic Simulation

Gabriel Rădulescu

Petroleum – Gas University of Ploieşti, 39 Bucureşti Blvd., Ploieşti, ROMÂNIA
e-mail: gabriel.radulescu@upg-ploiesti.ro

Abstract

Modelling and simulation, as one of the most used tools in processes investigation, are successfully applied for road traffic dynamic studies. As shown in the open literature, such a system with complex behavior is characterized by strong interactions between traffic participants, transport infrastructure and traffic controls, having a serious environmental impact – even deeper than other fields of human activity [1]. This paper addresses a modern modelling approach, originally adapted and included in the already announced software framework for controlled traffic investigation [3], as mathematical core-engine for independent lanes dynamic behavior description.

Key words: *road traffic, dynamic modelling.*

Introduction

Recently, the author of this paper has started a research project focused on road traffic mathematical modeling techniques, embedded within a modern framework which allows an easy traffic simulators implementation – presented in [3]. As the cited paper presents the project overview, from general aspects (like general/standard modeling and simulation approaches) to specific solved problems when building-up the software framework, this work offers a more complex look inside the mathematical model which is the core-engine of the application.

Modeling the traffic actors – a new approach

Since each mobile entity acts accordingly with its neighbors' behavior and (own) established rules, this work adopts a microscopic representation technique which may become the mathematical core of a traffic cellular automaton. This approach naturally leads to a significant flexibility in numerically defining a wide range of behavioral entities, which can be easily used for simulation and/or analysis purposes [2].

In the current representation, an independent traffic actor is determined by its *passive properties* (seen as model constant parameters: car length l, maximal acceleration a_+, maximal deceleration a_-, driver reaction time t_{react} and sensitivity S) and *active properties* (its allocated state variables: position x, actual speed v and acceleration a – updated with each simulation step Δt). Acceleration is considered as the main state variable, because it strongly depends on the environment and, more, v and x can be easily calculated from a. The only macroscopic parameter, seen as *traffic scene property*, is the maximum allowed speed value v_{max}.

Figure 1 shows the simplest case of a one-way road with only two cars, where vehicle 1 (in the back) behavior is described via the proposed algorithm, while vehicle 2 (in the front) is controlled by directly specifying its acceleration values over the entire simulation time horizon.

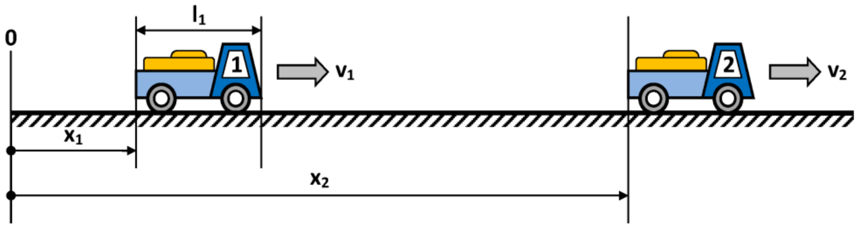


Fig. 1. Traffic scene: one-way road with two vehicles.

So, focusing on car 1 only, the first simplifying assumption is to have a constant acceleration value for each Δ*t* time horizon, the a_1 value (positive for acceleration, negative for deceleration) being directly influenced by the driver's actions on gas pedal. Considering also that a_1 should tend to its extreme values (a_{1+} or a_{1-}), the following equations in the model gives the vehicle 1 acceleration:

$$a_1(t) = \begin{cases} a_+ \tanh(S_1 \times \varepsilon_1(t)), & \text{if } \varepsilon \geq 0, \\ a_- \tanh(S_1 \times \varepsilon_1(t)), & \text{otherwise,} \end{cases} \tag{1}$$

where

$$\varepsilon_1(t) = \begin{cases} \Delta v_1(t) \times \left(x_2(t) - x_1(t) - l_1 - v_1(t) \times t_{1\,react} + \frac{\left(v_1(t) - v_2(t)\right)^2}{2a_{1-}} \right), & \text{if } \Delta dist_{12} < 0, \\ \Delta v_1(t) \times \left(x_2(t) - x_1(t) - l_1 - v_1(t) \times t_{1\,react} + \frac{\left(v_1(t) - v_2(t)\right)^2}{2a_{1+}} \right), & \text{otherwise.} \end{cases} \tag{2}$$

$\Delta dist_{12}$ represents the tendency of inter-vehicles distance variation, directly observed by driver 1. It takes into account the current time step (*t*) and the previous one ($t - \Delta t$), having negative values when $v_1 > v_2$ or non-negative values otherwise:

$$\Delta dist_{12} = \left(x_2(t) - x_1(t)\right) - \left(x_2(t - \Delta t) - x_1(t - \Delta t)\right). \tag{3}$$

$\Delta v_1(t)$ is the relative deviation between current vehicle 1 speed and its maximum allowed speed, v_{max}, calculated as

$$\Delta v_1(t) = \left(v_{max} - v_1(t)\right)/v_{max}\,. \tag{4}$$

Equation (1) establishes a direct dependency between acceleration *a* and *ε* which defines the deviation between *ideal* traffic conditions (free road, no maximum speed limit) and *real* ones. The author of this work propose a modified *ε* definition (in comparison with other classical approaches in the open literature – [2, 3]), which now *simultaneously* takes into account both restrictions (obstacles presence and speed limitations).

As shown in [3], for an independent traffic actor, *the fixed obstacles* (traffic lights, stopped cars) or *mobile* ones (moving vehicles on the same pathway) need a permanent state evaluation. But, regardless the obstacles type, the general *safety arrival distance* rule applies; it correlates the driver's actions (changes in *a*) with current traffic conditions, in a way allowing obstacles approaching, but never touching them. Considering vehicle 2 as the only (mobile) obstacle, the term $\left(x_2(t) - x_1(t) - l_1 - v_1(t) \times t_{1\,react} + \left(v_1(t) - v_2(t)\right)^2/2a_{1-}\right)$ in equation (2) estimates, at each time step, if car 1 can be safely slowed down when $\Delta dist_{12} < 0$ and a_1 hypothetically becomes a_{1-}. Greater this term is, safer its current situation becomes, while a negative value indicates the crashing danger; zero represents the critical limit when car 1 touches vehicle 2 exactly when v_1 becomes v_2 (so there will be no true collision after).

Of course, the same principle may be considered when evaluating the safety arrival distance rule for any fixed obstacle, $v_2(t)$ being replaced with zero in the term above, as shown in [3]. On the other hand, in equation (2) – after many experimental studies – the author of this work proposes a symmetric term in ε expression when $\Delta dist_{12} \geq 0$, finally leading to a true realistic vehicle behavior.

The speed limits, imposed by local traffic rules, road state and direction changes for instance, are taken into account by the term $\Delta v_1(t)$ in ε definition. Considering another simplifying assumption ($v_1(0) \leq v_{max}$, which is in fact absolutely normal), $\Delta v_1(t)$ is always positive and only slightly adjust the ε value when v_1 is close to v_{max}, until a_1 becomes zero. As time as the vehicle 1 speed value for the next step may be calculated with

$$v_1(t + \Delta t) = max(0, v_1(t) + a_1(t) \times \Delta t), \tag{5}$$

it is easy to demonstrate that, after several number of time steps Δt, v_1 will equal v_{max} whenever there is a safe distance between considered vehicles, proving a good adapting feature for the model (when new limitations – shown by changes in v_{max} – happen to occur). This approach can also be successfully applied to all dynamic changes in traffic regime, like traffic lights color switches and concurrence with vehicles having higher priority (when an additional decision structure completes the so-called *gap acceptance algorithm*) [2, 3].

Regarding the car 1 position, it is given by

$$x_1(t + \Delta t) = max\left(x_1(t),\ x_1(t) + v_1(t) \times \Delta t + \frac{a_1(t)\times(\Delta t)^2}{2}\right). \tag{6}$$

One can observe that equations (5) and (6) do not allow any negative values for v, respectively any x_1 decreasing tendency (meaning no turning back for the considered vehicle 1).

As remark, the positive or negative value of a is directly influenced only by ε, as all other terms in equation (1) are strictly greater than zero. Then, it can be observed that (1) brings a realistic representation of a depending on ε value by using the hyperbolic tangent operator, denoting a stronger driver's reaction on the gas pedal as the deviation (positive or negative) has a bigger absolute value [2].

Simulation results

For this paper, four simulation scenarios were selected, in order to prove the modified model adequacy in describing a two-vehicle traffic situation, where the car in front (2) is freely controlled (by directly specifying its acceleration $a_2(t)$ value(s) during simulation horizon, initial speed $v_2(0)$ and position $x_2(0)$), while the following car (1) behavior is modeled by the cinematic laws above presented. In all cases, vehicle 2 is characterized by $x_2(0)$ = 100m, $v_2(0) = v_{max}$ = 19.46m/s (70 km/h) and the same acceleration profile. Both vehicles have $a_{1+} = a_{2+}$ = 1.7m/s^2 and $a_{1-} = a_{2-}$ = –5m/s^2.

Scenario 1: $v_1(0)$ = 0m/s, sensitivity factor S = 2.5 (normal driving style)

Figure 2 presents how vehicle 1 reacts when starting with zero speed (at t = 0). The sensitivity factor value may be considered as medium/normal for this traffic case. First, the driver pushes completely the gas pedal ($a_1 = a_+$ = 1.7m/s^2) during the first 9 seconds. As consequence, v_1 rapidly increases from 0 to 16m/s, close to the maximum allowed speed v_{max} until (at $t \cong$ 10s), the brake is seriously hit ($a_1 \cong$ –3.2m/s^2) for a short time in order to prevent an imminent collision with vehicle 2. For the next 15s a_1 moderately increases, reaching again its maximum allowed value (1.7m/s^2) because there is no collision risk anymore. Since at t = 16s $v_1 = v_2$ = 3m/s, during the next time interval (t > 18s) it is expected that driver 1 will try to adapt its actions in order to keep v_1 as close as possible to v_2, maintaining this way an approximately constant safety gap ($x_2 - x_1$). One can see in figure 2 that the proposed algorithm successfully satisfies the *car following principle* above mentioned, for the chosen sensitivity value (2.5), the collision state being constantly kept at “0” (meaning car 1 never touches car 2, even when at t = 60s both vehicles are stopped).

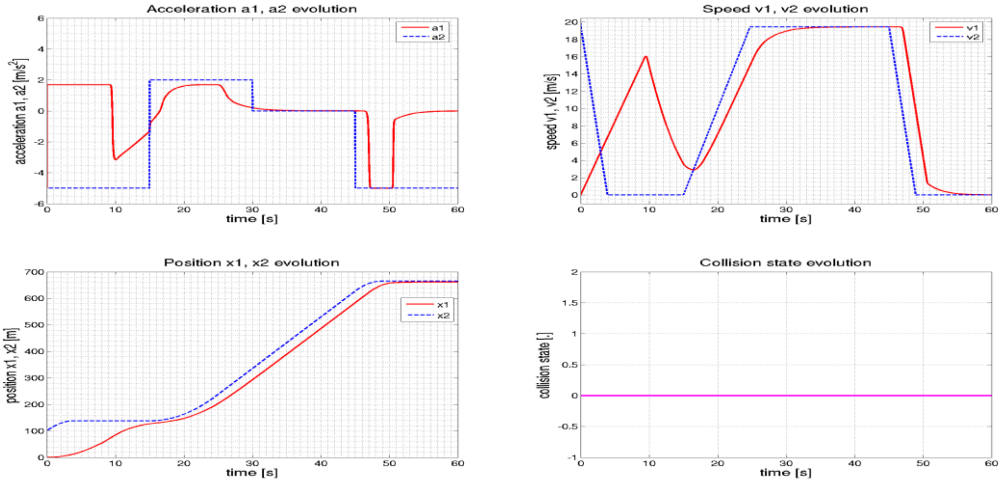


Fig. 2. Simulation results for scenario 1.

Scenario 2: $v_1(0)$ = 0m/s, sensitivity factor S = 0.1 (“lazy” driving style)

This new scenario differs from the first one only by intentionally considering a (very) low sensitivity factor value. As the good sense tells and figure 3 shows, the effect of a calmer action on the gas and brake pedals consists in a much slower speed variation, with lower amplitude (on corresponding time values) than in previous case. But, by analyzing the collision state evolution, it can be seen that vehicle 1 hits the car in front in two situations, at $t \cong$ 16s and $t \cong$ 51s (when collision state becomes “1”). In this case, the driver cannot keep a safe distance as it reacts too slowly when vehicle 1 suddenly stops (in about 4 seconds), because $a_2 = a_- = -5$m/s^2 at t = 0s and t = 45s. One can see in figure 3 how the v_1 profile is right-shifted from the previous case, meaning v_1 is adapted to v_2 with a serious delay, leading to this unwanted crashing situations.

Scenario 3: $v_1(0)$ = 0m/s, sensitivity factor S = 20.0 (“aggressive” driving style)

The third scenario illustrates the effect of a high sensitivity factor value, characterizing a sporty or nervous driver, on the controlled car (2) behavior. Such a driver over-estimates as potential dangers what all other drivers call “normal traffic situations” (i.e. a car in front quick speed decreasing, but still in the safe limits). On the other hand, the sporty/nervous driver usually hits the gas pedal shortly after he sees the distance to followed vehicle increases.

The proposed model successfully addresses this aggressive driving style simulation. As figure 4 depicts, by keeping the same behavior for vehicle 2, as well as other parameters for car 1 controlling algorithm (except the sensitivity factor), two false-critical time intervals can be identified (at t = 15s, for one second, and at t = 52s, for about 8 seconds), when vehicle 1 seriously approaches car 2. During these periods, driver 1 seems to nervously hit the brake, until it appreciates the “critical” situation ended. As remark, figure 4 presents only the acceleration evolution (with a zoomed vicinity of t = 16s), because all other diagrams look identical.

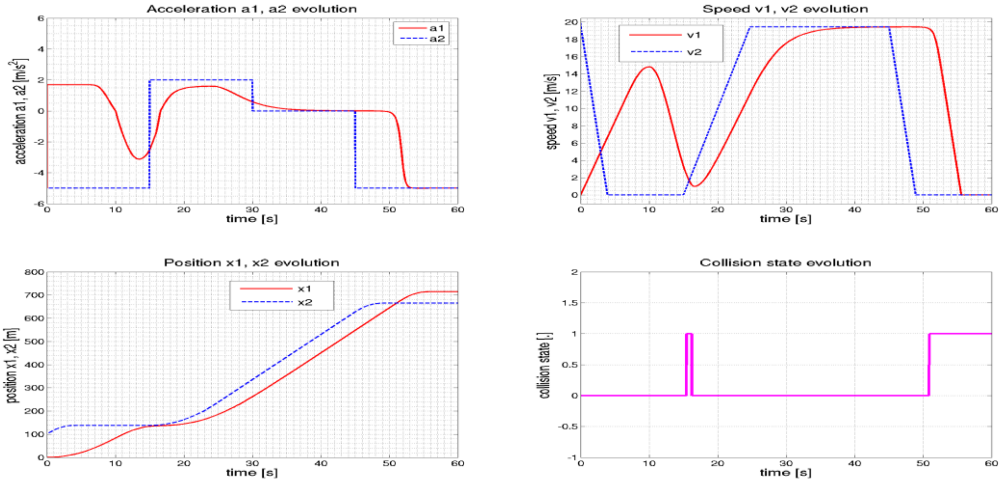


Fig. 3. Simulation results for scenario 2.

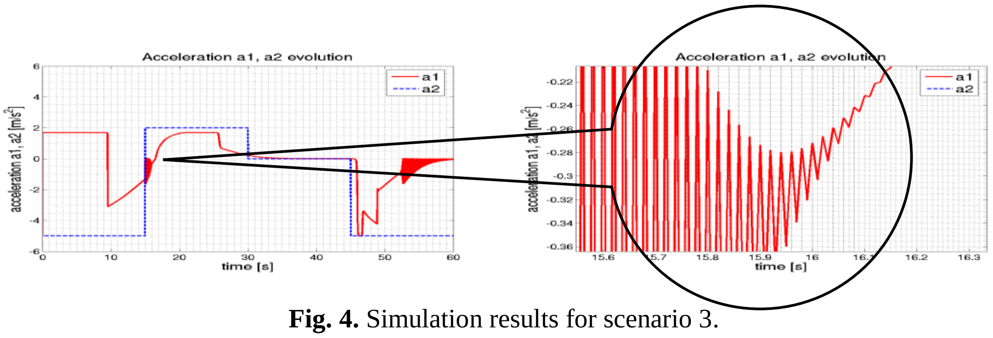


Fig. 4. Simulation results for scenario 3.

Scenario 4: $v_1(0)$ = 19.46m/s, sensitivity factor S = 10.0 (increased sensitivity)

Last chosen scenario represents another traffic situation, when vehicle 1 initial speed ($v_1(0)$) has the maximum allowed value, 19.46m/s, being the same as $v_2(0)$.

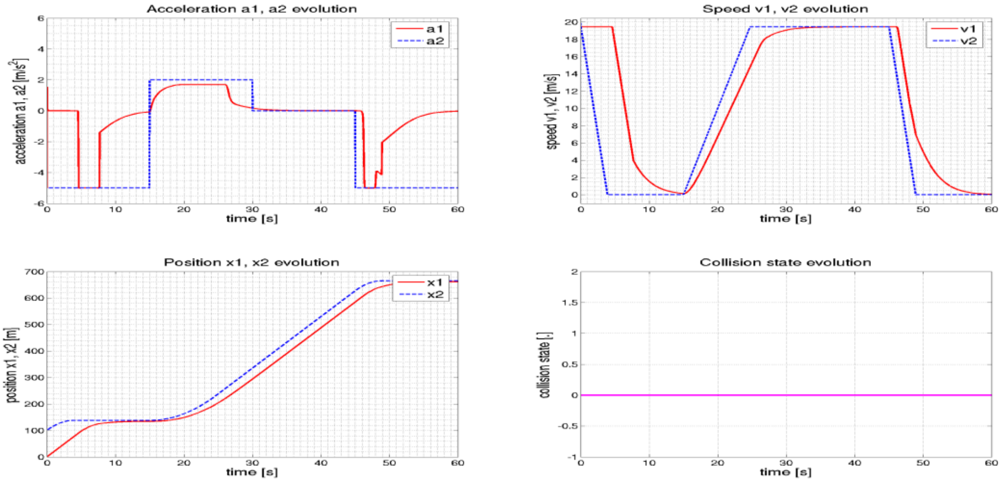


Fig. 5. Simulation results for scenario 4.

Although the results are not presented here, a sensitivity factor of 2.5 (like in scenario 1) proved not to be adequate anymore, as the high initial value of v_1 combined with a stiff situation ($a_2(0) = a_{–} = –3.2\text{m/s}^2$) imposes a different driver 1 attitude in order to slow down the vehicle within a safe time interval (meaning $x_2 – x_1 \geq l_1$ when $v_1 = v_2$). A test sensitivity value of 10.0 was used instead, the results depicted by figure 5 showing no collision for the entire simulation horizon.

Conclusions

This paper offers a more complex image on the mathematical model as the core-engine of a modern software framework (previously announced in [3]) allowing an easy traffic simulators design and implementation. Two changes in the model (introducing driver's sensitivity factor and fine acceleration tuning when approaching the maximum legal speed) were tested through simulation, with extremely promising results. In future research, the sensitivity must not have a constant value (as it is now), because traffic conditions are subject to serious variations from one scenario to another. The author will try to find an adaptive variation law for the sensitivity factor, where the main idea is to increase/decrease it until car in the back approaches the front car, and then revert it to a standard value (i.e. something between 2.5 and 10.0). As starting example, scenario 4 has to be considered: when t > 16s, sensitivity may be decreased because both vehicles start again with $v_1 = v_2 = 0$, somewhere at about 140m from the x-axis origin.

References

1. Roş, O., Gyenge Cs., Frăţilă, D. – *Sustainable Product Development by Considering the Environmental Consequences*. Proceedings of the 18th International DAAAM Symposium "Intelligent Manufacturing & Automation: Focus on Creativity, Responsibility and Ethics of Engineers", Zadar, Croatia, 24-27th of October, 2007.

2. Queloz, P.-A. – *Modèle de trafic routier et simulateur massivement parallèle, Master dissertation*. University of Geneva, 1995.

3. Rădulescu, G., Cruceru, M. – *A Modern Framework for Road Traffic Modelling and Dynamic Simulation*. Buletinul Universității Petrol-Gaze din Ploiești, Vol LIX, Seria Tehnică, No. 3/2008.

4. Yukawa, S., Kikuchi, M., Tadaki, S. – *Dynamical Phase Transition in One Dimensional Traffic Flow Model with Blockage*. Journal of the Physical Society of Japan, 63, 1994.

5. * * * – *Traffic Modeling – Phantom Traffic Jams and Traveling Jamitons*. Massachusetts Institute of Technology (http://math.mit.edu/projects/traffic/), 2009.

Un model matematic adaptat
pentru simularea fidelă a traficului auto

Rezumat

Modelarea și simularea, ca instrumente puternice asociate studiului sistemelor complexe, sunt aplicate cu succes în investigarea dinamicii traficului auto. Așa cum literatura menționează, un astfel de proces este caracterizat de puternice interacțiuni între entitățile participante, infrastructura rutieră și regulile de gestiune a circulației, având și un deosebit impact asupra mediului (ce poate depăși depășind chiar pe cel al industriilor productive [1]). Această lucrare prezintă o abordare modernă și originală a modelării matematice a traficului, ce se constituie în motorul platformei de simulare descrisă în lucrarea [3].