Description of the Matlab simulation framework

1 Functional description

This simulation framework makes it possible to investigate the behavior of a large number of systems within Matlab. It consists of a simulator core and a collection of blocks that implement the models to be simulated. This document describes only the simulator core as well as the formal block structure. The implementation of concrete blocks typically takes place in the context of the respective simulation application.

The primary area of application of this simulation framework is robotics. However, since the functionality is essentially determined by the available blocks, in principle any technical system can also be handled, for which suitable model descriptions are available and can be implemented with Matlab.

1.1 Overview

Figure 1 Figure 1 shows the essential elements of the simulation framework as well as their interaction. The simulator requires an experiment description in the form of a nested Matlab structure (struct). The simulator is called by one of the following commands:

```
simulate(<experiment-struct>);
results = simulate_unattended(<experiment-struct>);
```

The first variant starts the simulator including the visualization with full graphical user interface. The second command starts a basic version without visualization and is therefore suitable for integration into own scripts. Since this command is only a variation of the first command with reduced functionality, the following description refers only to the first variant.

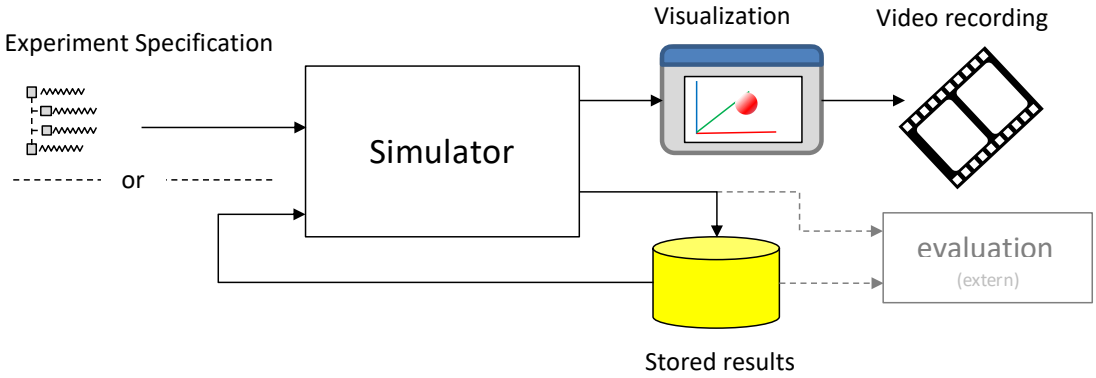


Figure 1: Elements of the simulation framework

Already calculated simulation results are visualized during the simulation. The type of visualization is defined by the blocks involved in the experiment. If necessary, the simulation can be decelerated to real time with configurable simulation time in order to realize a meaningful representation. If the function `simulate_unattended` (without visualization) is used, no speed reduction is performed and thus no extension of the simulation time. If the model is too complex for a real-time representation, the visualization will be delayed accordingly. In this case, no significant reduction of the simulation time can be achieved by using `simulate_unattended`.

All data generated during the simulation can be stored in a *.mat file as required and are therefore available for later evaluation. However, this evaluation is no longer part of the simulation framework described here. In addition, stored simulation data can be used instead of an experiment specification to start the simulator. In this case, the path to the * .mat file must be specified in the above function calls. In this way, it is possible to continue a previously interrupted simulation or to visualize already calculated results (possibly from a different perspective). The simulator core skips all calculations for existing data and executes only the visualization code. This is particularly useful for demonstrating the results, especially in the case of very slow model computing. In the case of very complex graphical representations, however, the visualization code can also lead to a significant delay so that a representation in real time is no longer possible. For these cases, it is possible to create a video recording from the main window of the visualization.

1.1 System structure

The main component of the experiment specification in Figure 1 is the system to be simulated. This consists of several blocks. These have an output and any number of inputs, to which the outputs of other blocks are connected. An example system of five blocks is sketched in Figure 2. The blocks connected to an output are referred to as "dependent blocks" because they require the (output) data provided by the preceding block as (input) data for their own calculation. Let $Succ(i)$ donate the set of dependencies w.r.t. the block i. The reverse relation, i.e. the set of blocks that provide input data for a block j, is $Pred(j)$. In the figure, e.g. the blocks 2 and 3 depend on block 1, i.e. $Succ(1) = \{2, 3\}$ and $Pred(2) = Pred(3) = \{1\}$. Block 1 is dependent on blocks 4 and 5. All connections are always regarded as non-reactive. There are no other additional requirements for the data format. Any Matlab-supported format can be used (scalar values, matrices, structures, cell arrays, interleaved structures...). If necessary, the format can be changed in each calculation step ("nonuniform output"). This, however, causes a larger overhead for saving the simulation results.

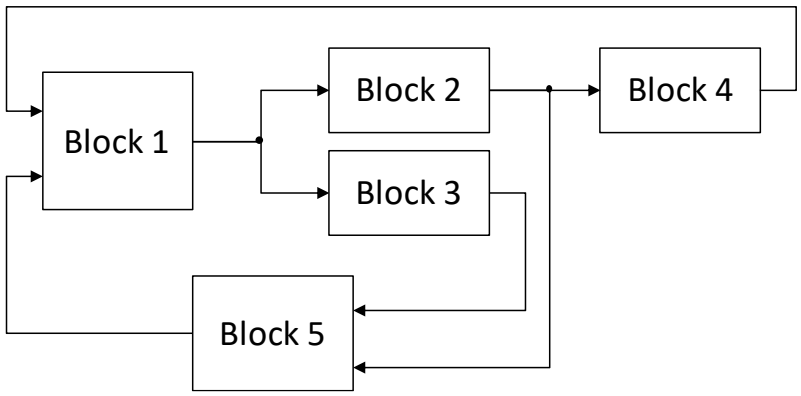
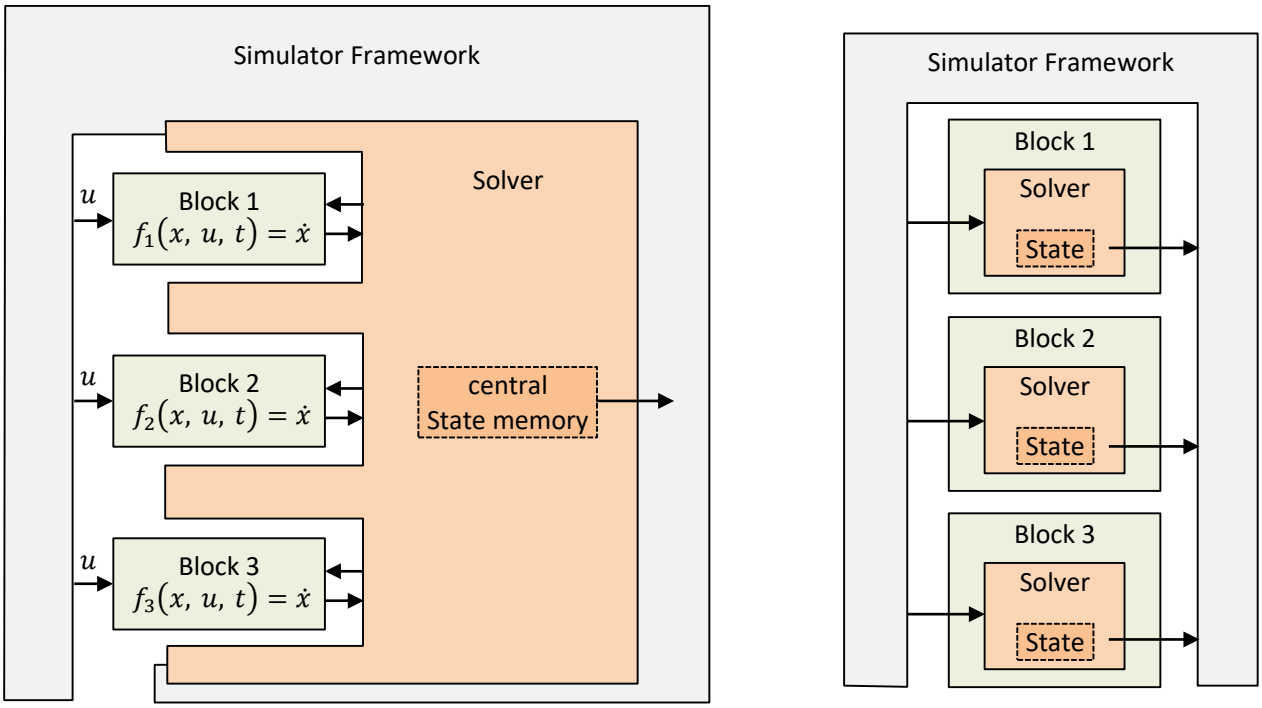


Figure 2: An example system consisting of five blocks

Structurally there is a strong similarity to Simulink. An advantage of the simulation framework described here over Simulink is the simplified visualization and data recording. The main difference, however, is the displacement of the solver from the simulator core into the model blocks. While the numerical solver is a central component of Simulink and each (time-continuous) block must be formulated in a compatible description, the simulation framework described here leaves the solution of model equations entirely to the respective blocks. (see. Fig. 3). The interface between the block and the simulator core is reduced to the return of the internal state and derived output data at certain, monotonously increasing instants. This is possible with static or time-discrete blocks without solvers. In continuous blocks, the desired status information can be determined, depending on the model complexity, analytically or by a suitable numerical technique.



(a) Architecture of Simulink

(b) Architecture of Simulation framework

Figure 3: compare of the Architecture

The chosen architecture may be less suitable for systems described primarily by time-continuous models. Simulink should continue to be used for such systems. The typical application for the solution presented here is the testing of complex signal processing algorithms. These are integrated into blocks operating in a time-discrete manner and, if necessary, coupled with a few continuous blocks which represent physical system components. A realistic example of a suitable system is the aircraft control system shown in Figure 4 with optically supported navigation. The flight dynamics is here the only time-continuous block ($\Delta t = 0$). The implementation can be carried out, for example, using the function *ode45*. The "sampling times" are determined by the connected time-discrete sensor blocks which are calculated periodically according to the defined sampling intervals T_{IMU} and T_{CAM}. The evaluation of the sensor data is carried out in complex algorithms, each of them is embedded in time discrete blocks. In the case of image processing, a trigger signal of the camera model controls the calculation times, whereby this block is automatically synchronized with the camera model.

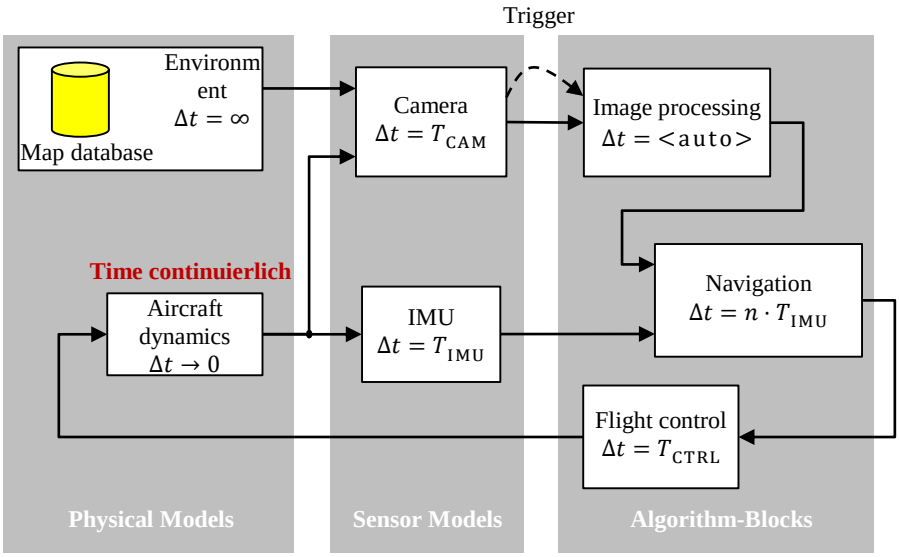


Figure 4: Aircraft control with optically supported navigation as an example system

1.2 How the simulation works

The simulation of a system is carried out by the repeated calculation of the constituent model blocks at specific discrete time points corresponding to the sampling intervals stored in the blocks. The generated output data are made available to the dependent blocks as input data. For this purpose, the simulation core maintains the current system time t and for block$i$the time of the last calculation $t_{i,last} \leq t$ as well as the next calculation time $t_{i,next} \geq t > t_{i,last}$. The simulation procedure is summarized in the following pseudocode formulation:

```
simulate(system)
    initialize: t = 0; ∀i t_i,last = −∞, t_i,next = t_i,1
    while experiment not finished do
        i_next = argmin_i(t_i,next)
        t = min(t_i,next)
        handle_block(i_next, i_next)
    end
end

handle_block(i, i_stop)
    foreach j ∈ Pred(i) do
       if t_i,last < t and t_i,next ≥ t and j ≠ i_stop
            handle_block(j, i_stop)
       end
    end
    process(i) // compute block outputs for time t
    t_i,last := t
    t_i,next := t + Δt_i
end
```

At the beginning, the simulation time is initialized to $t = 0$ and all $t_{i,next}$ are initialized with a user defined offset $t_{i,1} \geq 0$ (typ. $t_{i,1} = 0$). In the subsequent simulator main loop, the block i_{next} to be executed next is firstly determined using the smallest $t_{i,next}$-value. Before execution of this block, all blocks $j \in Pred(i_{next})$ are calculated with $t_{j,last} < t$ and $t_{j,next} \geq t$ to provide the necessary input data for i_{next}. This check is also performed recursively for all $k \in Pred(j)$ etc. If algebraic loops occur, they are separated at i_{next}. In general, the time $t_{i,next}$ will be the same for several or even all

blocks, so the selection is not unique. Due to the previously described dependency analysis, the concrete selection is however irrelevant for the calculation order (exceptions for algebraic loops!).

In the above illustration, specific details have not been given for the treatment of time-continuous blocks and the realization of trigger events.

1.2 Visualization and simulation control

Most of the visualization is in the main window of the simulation environment. This also includes the controls for simulation control. Figure 5 shows a screenshot of the main window with the most important elements.

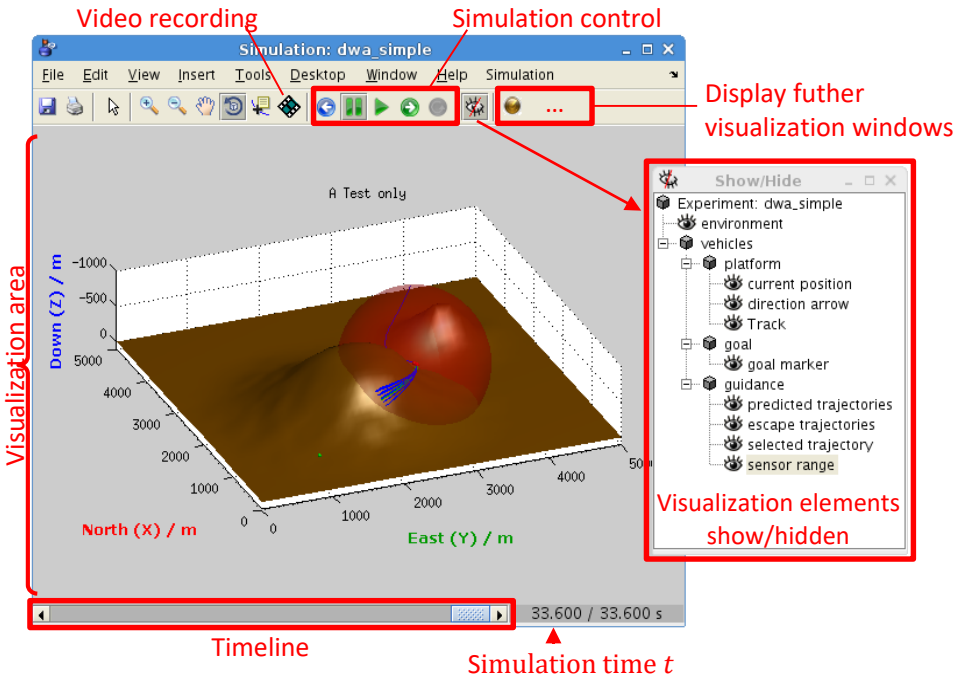


Figure 5: Screenshot of the simulation interface

The elements for simulation control in the toolbar enable the start / stop / resume of the simulation as well as the single step operation. Dieseling elements also control the replay function of already calculated data. The color of the symbols changes from green to blue. The visualization area consists of a Matlab axes. A variety of display parameters can be stored in the experiment structure, so that a two-dimensional or three-dimensional representation is produced according to the user's requirements. Within the visualization area, each block can place arbitrary graphics objects (Matlab Handle Graphics Objects). The simulation framework manages the handles and, if desired, displays them in a tree structure, so that individual objects can be selectively blended in and out at runtime in order to gain a better understanding of the system presented. The hiding of graphic elements also reduces the computational effort since the visualization code is only called for visible graphic elements. For complex model blocks, it can also be useful to visualize internal data outside the main window. Therefore, each block can define its own visualization window. These can be displayed and hidden using the icons on the right side of the toolbar.

Below the visualization area is the time bar which, like in a video player, allows a quick navigation through the already calculated simulation data. Next to this, the current simulation time as well as the latest time for which simulation data are already available are shown. Moving the time bar while the simulation is running pauses it automatically and switches to the replay mode (symbols become blue). When the time bar reaches the right edge in replay mode, the simulation continues seamlessly (symbols become green) and further data sets are calculated.