Causal Block Diagram assignment

Fall Term 2002

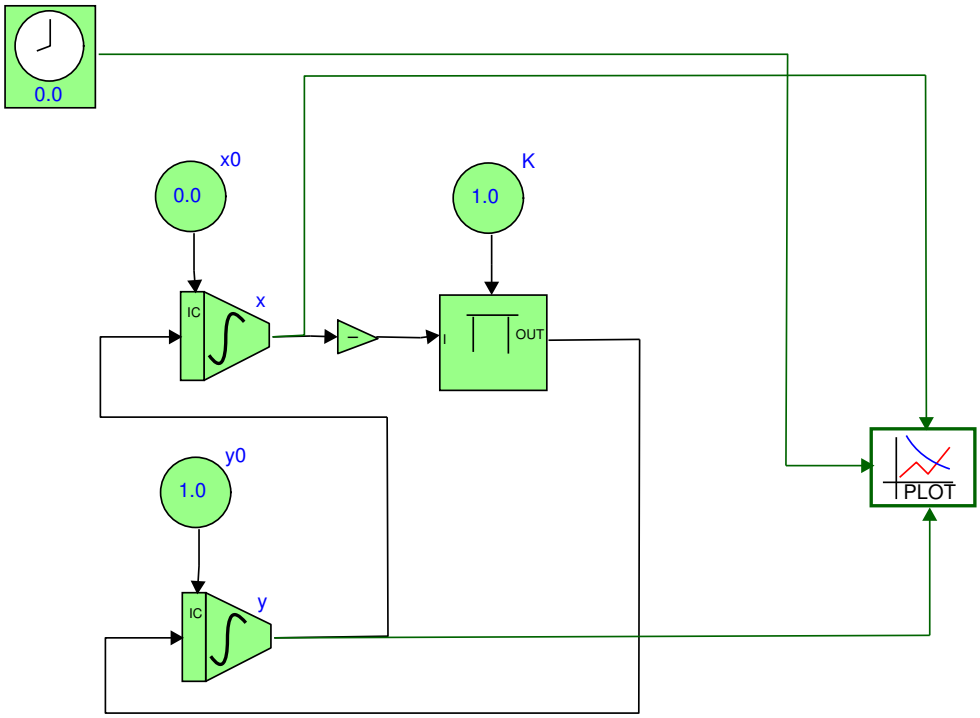
General Information

- The due date is **Wednesday 2 October 2002**, before midnight.
- Submissions must be done via WebCT. Beware that WebCT's clock may differ slightly from yours. As described on the Assignments page, *all* results must be uploaded to WebCT and accessible from links in the index.html file. There is no need to upload AToM3.
- The assignment can be made in groups of upto 3 people. It is understood that all partners will understand the complete assignment (and will be able to answer questions about it).
- Grading will be done based on correctness and completeness of the solution. Do not forget to document your requirements, assumptions, design, implementation *and* modelling and simulation results in detail !
- Extensions, if given, will involve extending not only the alotted time, but also the assignment.

The assignment

You will use the meta-modelling environment AToM^3 AToM3.asgn.tgz to interactively construct two Causal Block Diagram models:

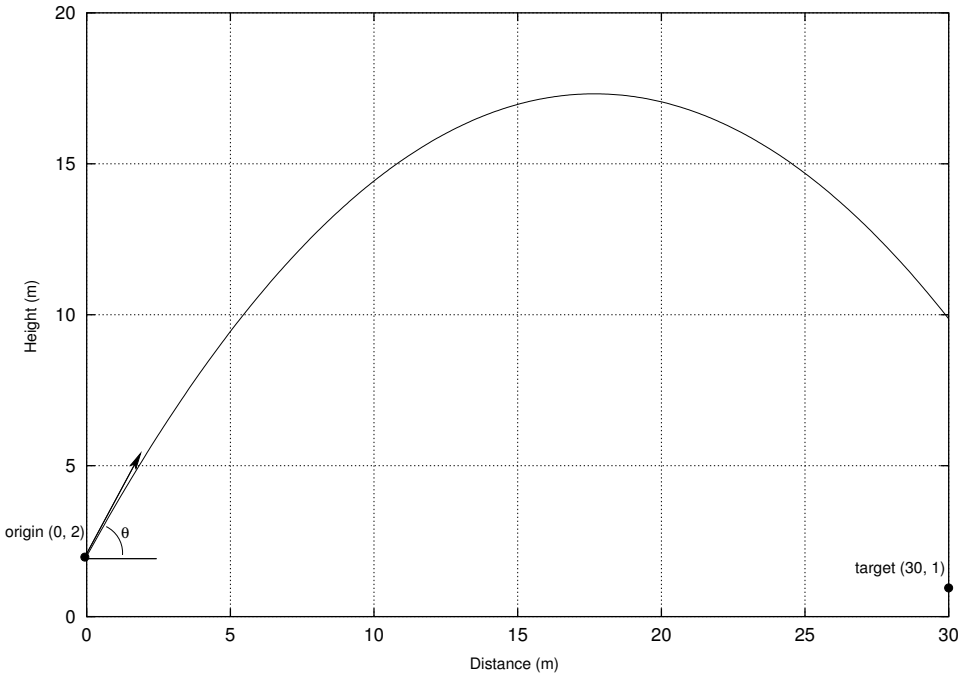
1. The "circle test" model (already given in the directory `ModelExpCircletest`).



2. The "ballistic" model (to be constructed and put in the directory `ModelExpBallistic`). This model must describe the trajectory of a football thrown from a height of $2m$ with an initial velocity v_0 of $20ms^{-1}$ at an angle θ with the surface of the earth. A host of simplifications apply:

   - The gravitation effect is considered constant with $g = 9.82ms^{-2}$.
   - The earth is considered flat near the ball thrower.
   - The ball's motion is abstracted to that of its centre of mass.
   - Friction (drag) is neglected.
   - ...

   Different values for $\theta \in [0, \pi/2]$ will result in different trajectories.



The assignment consists of four parts, some of which are optional.

1. Build and document Causal Block Diagram models for the circle test and the ballistic problems described above (the first is already done, but you're welcome to modify it).

2. (optional) Implement `EXPORT_LaTeX.py` which will export a LaTeXfile giving the "denotational semantics" of a model when the appropriate button is pressed in the modelling environment. To obtain for example

$$\begin{cases} \frac{dx}{dt} = y & x(0) = 0 \\ \frac{dy}{dt} = -Kx & y(0) = 1, K = 1 \end{cases}$$

   from the circle test model, the model's data structure must be traversed and the following written to file:

```
\documentclass[12pt]{article}
\begin{document}
\[
 \left\{
  \begin{array}{ll}
   \frac{dx}{dt} =   y & x(0) = 0\\
   \frac{dy}{dt} = -Kx & y(0) = 1, K=1
```

```
  \end{array}
 \right.
\]
\end{document}
```

3. (optional if you're working alone) Implement `EXPORT_Mfile.py` which will produce a representation of the model suitable for processing by a tool such as Matlab/Simulink (www.matlab.com) or a public version such as Octave www.octave.org. The following gives the circle test model in a format suitable for Octave:

```
function xdot=f(x,t)
 xdot=zeros(2,1);
 xdot(1)=x(2);
 xdot(2)=-x(1);
endfunction
x0=[0;1];
```

   To obtain a solution and plot it in Octave:

```
t=linspace(0,50,1000);
y=lsode("f", x0, t);
plot (t,y);
plot (y(:,1), y(:,2))
```

   A similar syntax is used by Matlab. Demonstrate the working of your exporter by applying it at least to the circle and ballistic tests.

4. (a) Implement (in the files `MODEL_plotWindow.py, SIM_pause.py, SIM_reset.py, SIM_startResume.py`) a Time Slicing simulator. This includes algebraic loop detection, topological sorting of blocks, and the actual time slicing.
   (b) Demonstrate the correct working on the circle test (with a small and with a large Δt stepsize).
   (c) Perform a series of simulation experiments to determine the optimal throwing angle θ if one wants to hit a target at $30m$ distance and $1m$ height as fast as possible for the given v_0.

Derivation of the Ballistic motion ODE

The following derives the equations of motion for the ballistic problem above. You have to convert these into a Causal Block Diagram.

One of our assumption states that the ball's motion is abstracted to that of its centre of mass. Hence we start from Newton's Second Law of motion:

$$\mathbf{F} = m \cdot \mathbf{a},$$

where the force $\mathbf{F}$ and the acceleration $\mathbf{a}$ are both vectors. In the plane, we thus have

$$\begin{cases} F_x = m \cdot a_x \\ F_y = m \cdot a_y. \end{cases} \tag{1}$$

The only force that influences our football is the gravitational force, hence $F_x = 0$ and $F_y = -m \cdot g$, with g the gravitational acceleration ($9.81\ m/s^2$ in our problem). Substituting this in equation 1, we get

$$\begin{cases} a_x = 0 \\ a_y = -g. \end{cases} \tag{2}$$

Recall that the acceleration is the second derivative of the position with respect to time: hence $a_x = \ddot{x}$, $a_y = \ddot{y}$ (a dot denotes a derivative with respect to time). We now have a system of two *second-order* equations. Since most ODE solvers (including our Time Slicing simulator) operate on a system of *first-order* equations, we use the standard technique of introducing extra variables: we let $v_x = \dot{x}$, and $v_y = \dot{y}$ (respectively, the horizontal and vertical velocities). We then have the following system to solve:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{v}_x = 0 \\ \dot{v}_y = -g \end{cases} \tag{3}$$

The above model describes a *class* of behaviours. To specify a unique solution trajectory (our Time Slicing simulator is not capable of producing a class of solutions as a symbolic solver is), we need to specify initial conditions. The initial conditions for each of the variables in equation 3 should be easy to determine from the problem specification (recall that the given v_0 is the magnitude of a velocity *vector* $\mathbf{v_0}$.

Using the AToM^3 environment

Download AToM3.asgn.tgz archive.

Expand it locally (for example with the command `tar --ungzip -xvof AToM3.asgn.tgz` if you're working on a UNIX machine. This will create a directory `AToM3.asgn`.

To start the AToM^3 environment, `python ATOM3.py`. `python` should be at least version 2.2 of Python (this is installed in the SOCS labs). On UNIX, the script `atom3` is a shortcut for the above command. On windows, you can just click on the ATOM3 icon to launch it.

AToM^3 will be started with the CausalBlockDiagram formalism loaded. You can either build your own model (and `File/Save` it) or `File/open` an existing one. In the directory `ModelExpCircletest` you will find the example model `circle_CausalBlockDiagram_mdl.py`.

A directory `ModelExpBallistic` has been set up for your ballistic model and experiments.

When AToM^3 starts with the CausalBlockDiagram formalism, it will use a number of files in the `CausalBlockDiagram` directory:

- `MODEL_plotWindow.py` contains code which gets called when you click on a Plot icon. It opens a plot window. When pressing "new" in that window, it is possible to add plot items (one variable as a function of another).
- `EXPORT_LaTeX.py` constains code which gets called when you press the "to LaTeX" button. Currently, it contains some code which demonstrates how to traverse the model data structure. With this example, you should have enough information to complete the assignment.
- `EXPORT_Mfile.py` contains code which gets called when you press the "to Matlab/Octave" button.
- `SIM_startResume.py` contains the startResume(parentApp, model) method which gets called when you press the "Exp Start/Resume" button. Currently it starts a thread which contains a data generator. You have to replace this generator by a Time Slicing simulator which produces data using the model structure and node attributes as well as the global model attributes `simul_t_init`, the simulation start time, `simul_t_final`, the simulation end time, and `simul_delta_t`, the simulation step-size.
  Note how we implicitly assume the termination condition for the simulator is $t > t_{final}$. You may want to use another termination condition.
- `SIM_pause.py` and contains code which pauses the simulation. It gets called when the "Exp Pause" button is pressed. Note how this is not yet the case in the "starting point" code. Making "pause" work is a low priority.
- `SIM_reset.py` contains code which gets executed when the "Exp Reset" button is pressed. Currently, this clears the data set displayed by the plotter. In your Time Slicing simulator you should make sure properly resets the experiment to its initial state.

A pretty-printed version of these files (with `enscript --color -E -Whtml --toc -p pretty_print.html *.py`) is here.

Updates

- Monday 23 September:
  AToM3.asgn.tgz has been updated. It is still compatible with the models you have built upto now and the code you may have written.
  The archive now contains:

  1. A larger set of global model attributes: `model_name`, `model_author`, `model_creation_data`, `simul_t_init`, `simul_t_final`, `simul_delta_t`, `simul_comm_interval`, `user_string`, `user_float`. The example model `circle_CausalBlockDiagram_mdl.py` now has meaningful values for these attributes.
  2. The `Time` block now displays the `block_name`.
  3. A string attribute `block_operator` has been added to all blocks. It is only useful for the `Generic` block. You can use it to make custom operator blocks such as `sin` and `cos`. Your Time Slicing simulator will have to process these blocks appropriately. Don't forget to document which operators you will be supporting.
  4. In `SIM_startResume.py`, the variable names in the `sendPlotter` method were changed for improved readability.
  5. Button images have been updated.

  A clarification: the `block_out_value` attribute of a block must contain the output signal value at the "current" time. It must thus be updated by your Time Slicing simulator. As described in the CBD notes, you need temporary storage for these values while processing Delay/Integrator/Derivative blocks. This is provided for in the form of the `block_out_value` attribute (note how you could also just create a new attribute on the fly in Python).