Cyber-Physical Systems and Discrete Controllers

CS684: Embedded Systems

Topic 1

Paritosh Pandya

Indian Institute of Technology, Bombay

January 12, 2023

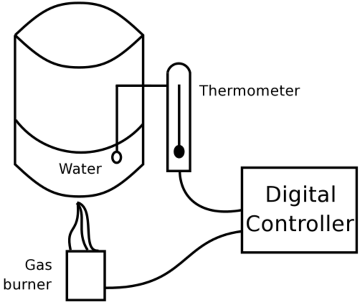
Cyber Physical System

- Physical system consisting of mechanical and electronics components controlled by computerised controller.
- Languages for Programming Discrete Controllers: C, SystemC, Verilog/VHDL, State charts, Simulink/Stateflow, Lustre/SCADE/Heptagon.

Synchronous Dataflow Programming

- Using language Lustre/Heptagon
- Compiler: Lustre/Heptagon $\rightarrow$ C/Verilog
- Why Lustre/Heptagon?
    - Very high level modelling language for Discrete Control
    - Functional Dataflow Programs
    - Logical concurrency with clean semantics
    - Determinsitic execution
    - Automatic generation of Efficient C code

Example: Cyber-Physical System



Example: Cyber-Physical System

- Behavior of the temperature in the tank
  - When the gas burner is OFF the temperature evolves according to
    $$x(t) = I e^{-Kt}$$
    i.e. $\dot{x} = -Kx$
  - When the gas burner is ON the temperature evolves according to
    $$x(t) = I e^{-Kt} + h(1 - e^{-Kt})$$
    i.e. $\dot{x} = K(h-x)$

  Where I is the initial temperature of the water, K is a constant that depends on the nature of the tank (how much it conducts heat for example), h is a constant that depends on the power of the gas burner, and t models time.

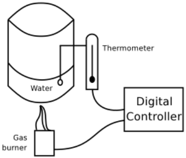
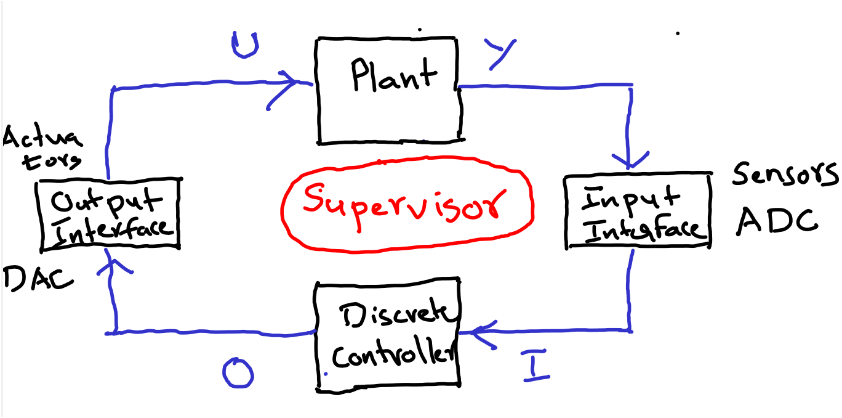


Fig. 1. Our running example

- We will refer to ON and OFF as modes of the tank evolution.

Model of Cyber-Physical System



Physical Plant

- Includes physical device and environment
- Continuous Dynamics
  Input U, Output Y and State of the system X.
  Continuous trajectory $X : \Re_0 \rightarrow Val_X$
- Mathamtical Model using Differntial Equations

$$
\begin{aligned}
X' &= G(X, U) \\
Y &= E(X, U)
\end{aligned}
$$

Input Interface

- Converts a contiuous time physical signal and gives its value to the discrete controller at discrete time points (sampling points).
- Example Analog to Digital Converter (ADC). Organized as Input Driver Library of API calls.
- Conversion and data transfer is done at discrete time points (sampling points) decided by the supervisor.
- General mathematical model of Interface: Combined Differential+Difference equations.

Output Interface

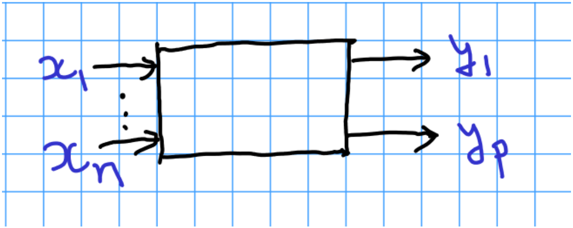
- Converts a discrete time signal from controller into contiuous physical signal tp the plant.
- Example Digital to Analog Converter (DAC). Some outputs may require further computation provided by output drive library.
- Conversion and data transfer is done at discrete time points (actuation points) decided by the supervisor.
- General model of Interace: Combined Differential+Difference equations.

Discrete Controller

- Invoked by supervisor repeatedly.
- At each invocation supervisor provides input value.
- Discrete controller produces coresponding output value.
- current output can depend on the sequence of past inputs.
- Example `node MINSOFAR( I : int) returns (O : int)` returns minimum of inputs seen so far.

I	4	0	2	-3
O	4	0	0	-3
clock	0	1	2	3 …

Discrete Conctroller Input Output



Discrete Controller

- Works in discrete steps of computation.
- Realised using computational devices like Microcontroller, or FPGA, or Sequential Circuit with Flipflops and Gates.
- Inputs I, Outputs O and State Z.
- Time is Discrete, i.e. $\mathbb{N}$. A time points is called a clock cycle, tick, or reaction. Discrete flow: $I : \mathbb{N} \rightarrow Val_I$

Discrete Controller

- Works in discrete steps of computation.
- Realised using computational devices like Microcontroller, or FPGA, or Sequential Circuit with Flipflops and Gates.
- Inputs I, Outputs O and State Z.
- Time is Discrete, i.e. $\mathbb{N}$. A time points is called a clock cycle, tick, or reaction. Discrete flow: $I : \mathbb{N} \rightarrow Val_I$

Model (Forward): Difference Equation / Recurrence

$$
\begin{aligned}
Z_0 &= R(I) \\
Z^+ &= F(Z, I) \\
O &= H(Z, I)
\end{aligned}
$$

Discrete Controller

Model (backward): Difference Equation / Recurrence.

$$
\begin{aligned}
Z_0 &= R(I) \\
Z &= F(Z^-, I) \\
O &= H(Z, I)
\end{aligned}
$$

Discrete Controller

Model (backward): Difference Equation / Recurrence.

$$
\begin{aligned}
Z_0 &= R(I) \\
Z &= F(Z^-, I) \\
O &= H(Z, I)
\end{aligned}
$$

Behaviour of controller

I	I_0	I_1	I_2	...
Z^-	$-$	Z_0	Z_3	...
Z	Z_0	Z_1	Z_2	...
O	O_0	O_1	O_2	...
clocktick	0	1	2	...

Example of Discrete Controller

A discrete controller which at each step returns the minimum of input values seen so far.

```
node MINSOFAR( I:int) returns (O:int)
var m:int;
```

$$
\begin{aligned}
m_0 &= I_0 \\
m_i &= min(m_{i-1}, I_i) \\
o_i &= m_i
\end{aligned}
$$

Behaviour

I	4	0	2	-3
M	4	0	0	-3
O	4	0	0	-3
$tick$	0	1	2	3 ...

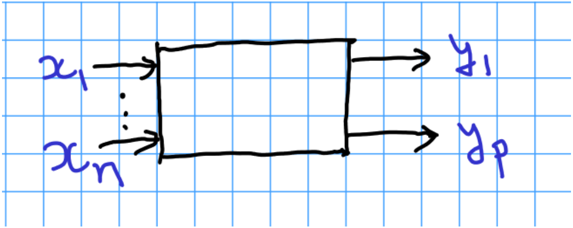
Typical Supervisors

Event and Time Triggered Systems

```
InitializeControllerMemory;
for each input event  do
   Read Inputs;
   Controller step{
     <Compute Outputs>
     <Update ControllerMemory>}
   Perform Actuation
end
```

```
InitializeControllerMemory;
for each clock tick  do
   Read Inputs;
   Controller step{
     <Compute Outputs>
     <Update ControllerMemory>}
   Perform Actuation
end
```

Reactive Kernel Input Output



Reactive Kernel

- Definition of Controller Memory data type `Controller_f_mem`
  Definition of Structure defining Output data type `Controller_f_out`
- Declaration of controller Memory variable `mem :Controller__f_mem`
- **Reset function** Procedure resetting controller to initial state Z_0.
  `void Controller__f_reset(Controller__f_mem* self)`
- **Step function** Procedure computing next memory state $+Z$ and current output O.

```
void Controller__f_step( t1 x1 , ..., tn xn ,
                         Controller__f_out* _out,
                         Controller__f_mem* self );
```

A Supervisor for Simulating Discrete Controller

```
#include "controller.h"
int main(int argc, char * argv[]) {
/* declare state variables */
  Controller__f_m mem;
/* declare input variable */
  t1 x1 ;
  ...
  tn xn ;
/* declare output variables */
  Controller__f_out ans;
```

Supervisor for Simulating Discrete Controller

```
/* initialize memory instance */
   Controller__f_reset(&mem);
/* repeatedly perform reaction cycle */
   while(1) {
      /* read inputs */
      scanf("...", &x1 , ..., &xn );
      /* perform step */
      Controller__f_step(x1 , ..., xn , &ans, &mem);
      /* write outputs */
      printf("...", ans.y1 , ..., ans.yp );
      }
```

Summary

- Cyber physical system consists of physical elements controlled by the conglomorate of Supervisor, Discrete Controller and the Input-output Interfaces.
- For interacting with physical devices, the supervisor uses API calls from Input Interface Drivers and Output Interface Drivers.
- Control is achived by repeated execution of Sense, ComputeStep, Actuate cycle as directed by the supervisor.
- A Discrete Controller transforms an input flow into an output flow, synchronously.
- A discrete controller can be given as reactive kernel consisting of a reset and a step function with associated declarations.

High Integrity Embedded Systems

- Examples Nuclear reactors, Air Craft Controller, Space Ships (Launch Vehicles, Landers), Defense equipment,yTransportation systems, Medical equipment, Smart ...
- Safety Critical and Mission critical Cost of failure high

Challanges

- Low productivity. High development costs. High time-to-market.
- Certification Software and its development lifecycle
  DO-178B (Aircrafts)
- Validation and Verification

Model Based Design of Discrete Controller

- Build a high level model of System (Discrete Controller) Behaviour
- Executable: Deterministic, concurrent, modular, hierarchical.
- Validation and Verification:
  - Simulation
  - Testing: Requirement based, Coverage.
  - Formal Verification
- Automatic code generation: certified.

Languages for Model Based Design

State Charts, Simulink+State-flow, Verilog, Lustre/Scade/Heptagon