Universiteit Antwerpen
Faculteit Wetenschappen

Modelling Of Software Intensive Systems

Assignment 2: CBC

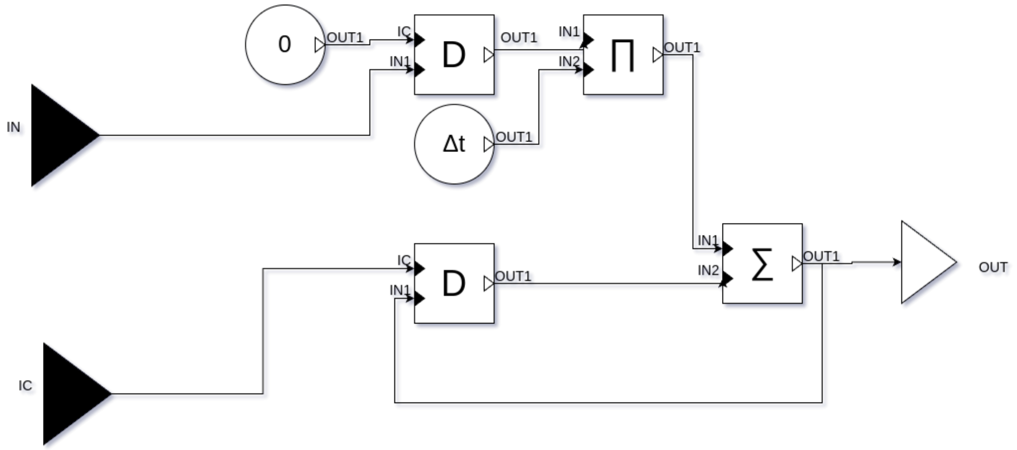
1st Master computer science
2024-2025

Liam Leirs
Robbe Teughels

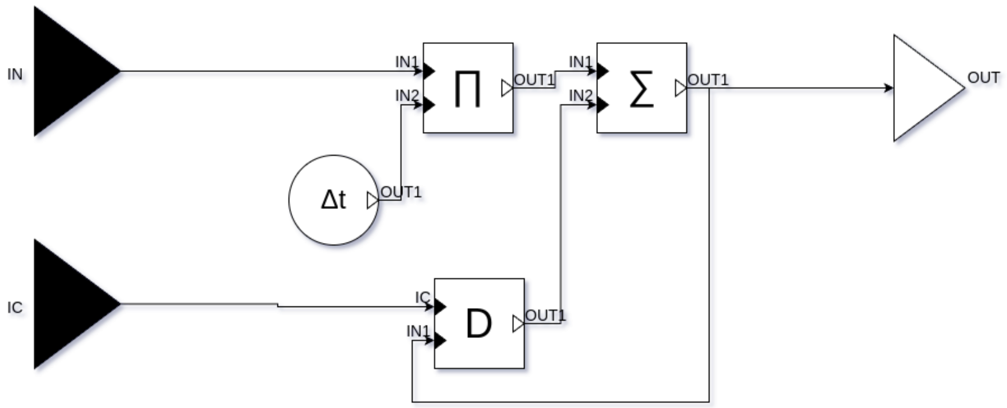
Integration Methods

For each integrator, we constructed a custom CBD block with an initial condition (IC) input. This initialization ensures that each integrator block starts from a defined value.

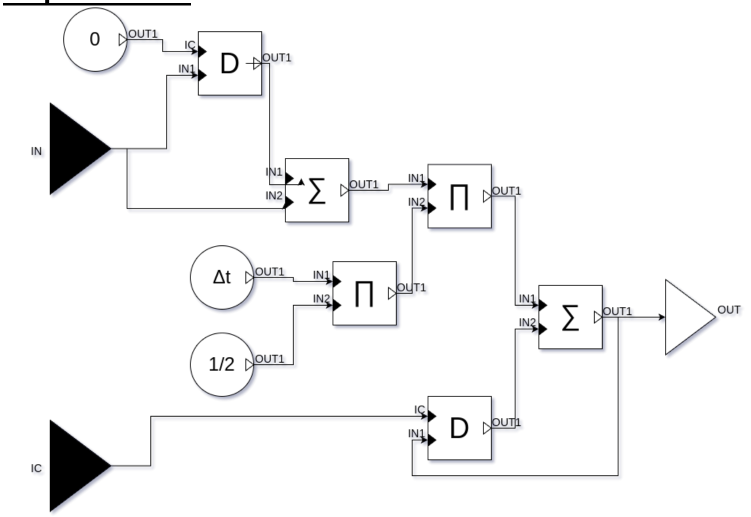
<u>Backwards euler</u>



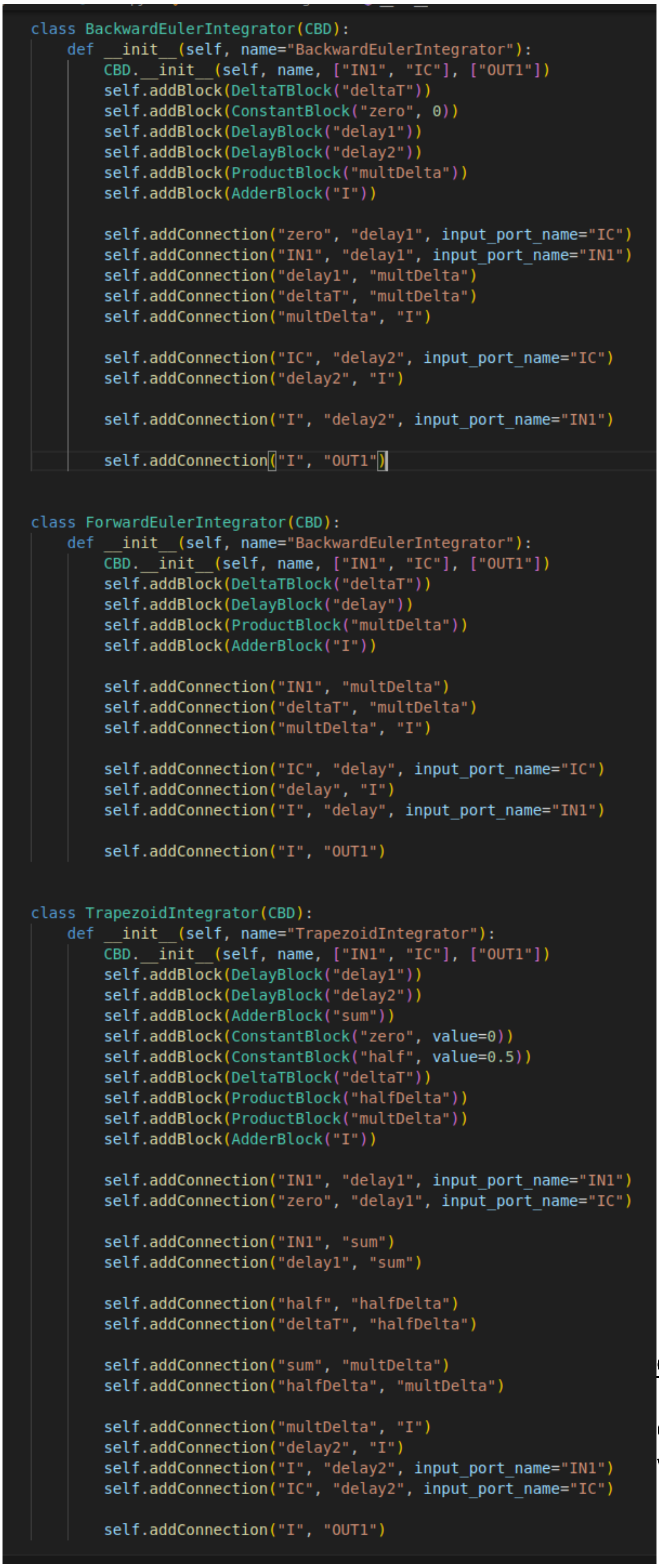
<u>Forwards euler</u>



<u>Trapezoid rule</u>



<u>Python code for blocks</u>



```
class BackwardEulerIntegrator(CBD):
    def __init__(self, name="BackwardEulerIntegrator"):
        CBD.__init__(self, name, ["IN1", "IC"], ["OUT1"])
        self.addBlock(DeltaTBlock("deltaT"))
        self.addBlock(ConstantBlock("zero", 0))
        self.addBlock(DelayBlock("delay1"))
        self.addBlock(DelayBlock("delay2"))
        self.addBlock(ProductBlock("multDelta"))
        self.addBlock(AdderBlock("I"))

        self.addConnection("zero", "delay1", input_port_name="IC")
        self.addConnection("IN1", "delay1", input_port_name="IN1")
        self.addConnection("delay1", "multDelta")
        self.addConnection("deltaT", "multDelta")
        self.addConnection("multDelta", "I")

        self.addConnection("IC", "delay2", input_port_name="IC")
        self.addConnection("delay2", "I")

        self.addConnection("I", "delay2", input_port_name="IN1")

        self.addConnection("I", "OUT1")


class ForwardEulerIntegrator(CBD):
    def __init__(self, name="BackwardEulerIntegrator"):
        CBD.__init__(self, name, ["IN1", "IC"], ["OUT1"])
        self.addBlock(DeltaTBlock("deltaT"))
        self.addBlock(DelayBlock("delay"))
        self.addBlock(ProductBlock("multDelta"))
        self.addBlock(AdderBlock("I"))

        self.addConnection("IN1", "multDelta")
        self.addConnection("deltaT", "multDelta")
        self.addConnection("multDelta", "I")

        self.addConnection("IC", "delay", input_port_name="IC")
        self.addConnection("delay", "I")
        self.addConnection("I", "delay", input_port_name="IN1")

        self.addConnection("I", "OUT1")


class TrapezoidIntegrator(CBD):
    def __init__(self, name="TrapezoidIntegrator"):
        CBD.__init__(self, name, ["IN1", "IC"], ["OUT1"])
        self.addBlock(DelayBlock("delay1"))
        self.addBlock(DelayBlock("delay2"))
        self.addBlock(AdderBlock("sum"))
        self.addBlock(ConstantBlock("zero", value=0))
        self.addBlock(ConstantBlock("half", value=0.5))
        self.addBlock(DeltaTBlock("deltaT"))
        self.addBlock(ProductBlock("halfDelta"))
        self.addBlock(ProductBlock("multDelta"))
        self.addBlock(AdderBlock("I"))

        self.addConnection("IN1", "delay1", input_port_name="IN1")
        self.addConnection("zero", "delay1", input_port_name="IC")

        self.addConnection("IN1", "sum")
        self.addConnection("delay1", "sum")

        self.addConnection("half", "halfDelta")
        self.addConnection("deltaT", "halfDelta")

        self.addConnection("sum", "multDelta")
        self.addConnection("halfDelta", "multDelta")

        self.addConnection("multDelta", "I")
        self.addConnection("delay2", "I")
        self.addConnection("I", "delay2", input_port_name="IN1")
        self.addConnection("IC", "delay2", input_port_name="IC")

        self.addConnection("I", "OUT1")
```

Comparison

Computing the integral for g(t) we get the following values:

Delta t	Backwards euler value	Forwards euler value	Trapezoid rule value
0.1	3.222190908877023	3.223153281886757	3.2226720953818915
0.01	3.213459296657502	3.213555450684053	3.213507373670786
0.001	3.212588796472303	3.212598411043006	3.21259360375765

Comparing with the analytical solution we get:

Delta t	Backwards euler error	Forwards euler error	Trapezoid rule error
0.1	0.009698804877023015	0.010661177886757134	0.010179991381891629
0.01	0.0009671926575021139	0.001063346684053279	0.0010152696707863562
0.001	9.669247230315037e-05	0.0001063070430062929	0.00010149975765028074

As delta t increases, the approximation gets closer and closer to the actual value. Here we can also see that the backwards euler method gives us the best approximation in this case.

Co-Simulation

Sub-task 1: Plant FMU from Modelica

For simulating the gantry system in an other environment without giving the model, we generate a FMU of the model. We need to make sure that all variables to evaluate the system are available to the outside of the FMU by creating output pins.

Sub-task 2: PID Controller in PyCBD

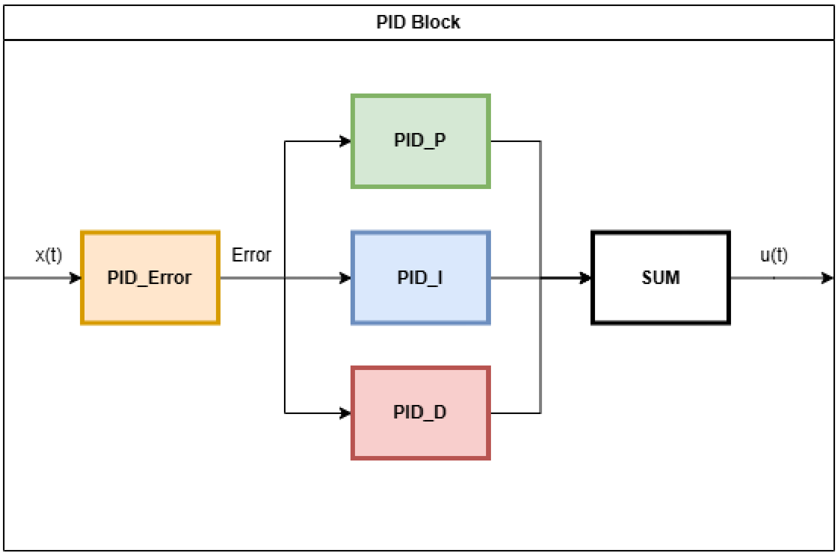
We want to create a similar PID controller as In the fist assignment using a different framework. The controller is defined as in the following representation. For simplicity and a better overview of connections and blocks viewed in lines, the functionality is split to distinct parts of the PID controller. PID_Error, this block takes the input value and returns the error value. This value gets directed to the PID_P, PID_I and PID_D. The output gets summed up resulting in u(t).

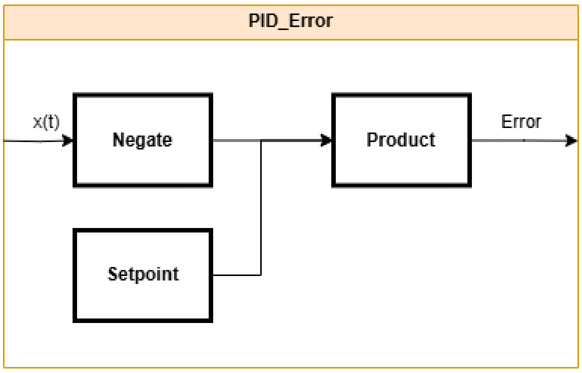
Sub-task 3: Controller FMU from Controller CBD

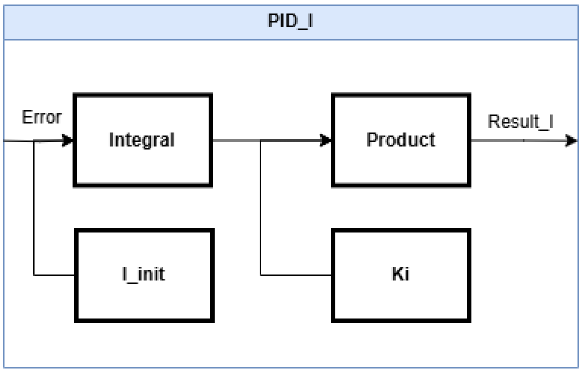
Generating a FMU of the controller, we observe modelDescription.xml and model.c. The modelDescription contains all variables defined inside the model, as follows: modelname.variablename.portname. For each of these variables, we can see if they are calculated, constants. Giving some detail without saying how it works. At the other hand, model.c contains all equasions, initial as runtime equations. These are used in a control loop to evaluate values and run the model.

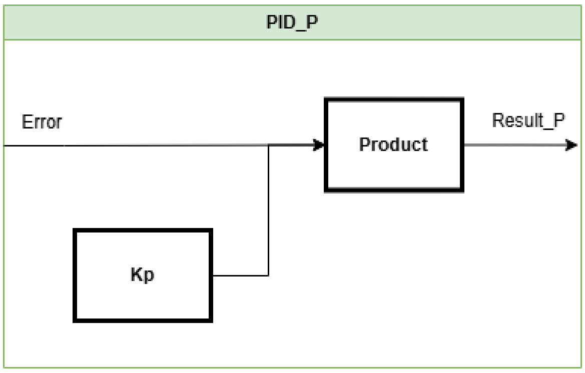
Sub-task 4: Compile Controller C-Code and Co-simulate

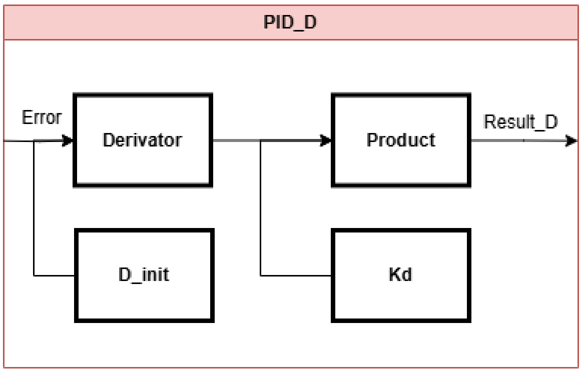
Using the provided script, we can co-simulate the controller and the plant. The results are given below. As the internal working of the PID in Modelica is exactly the same as the CBC in pyCBD, using the exact same parameters Kp, Ki and Kd results in the exact same set of equations and constants so also the same output. However, given that now Ki = 1, the result of u(t) will slightly vary from the modelica results.

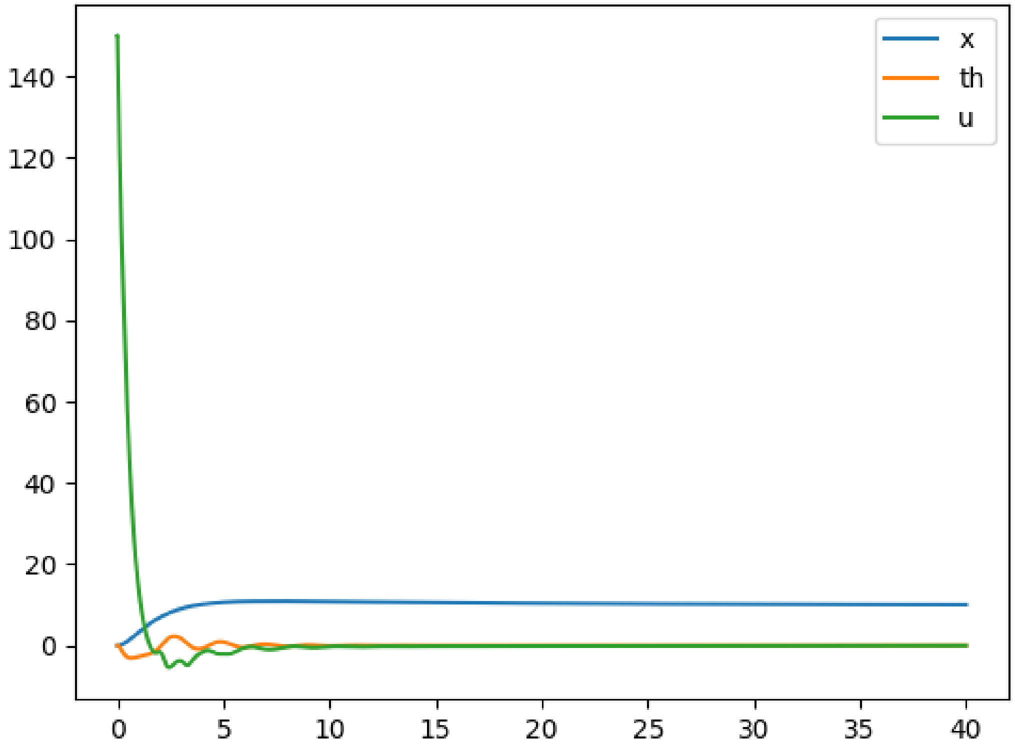

PID Block
x(t)
PID_Error
Error
PID_P
PID_I
PID_D
SUM
u(t)

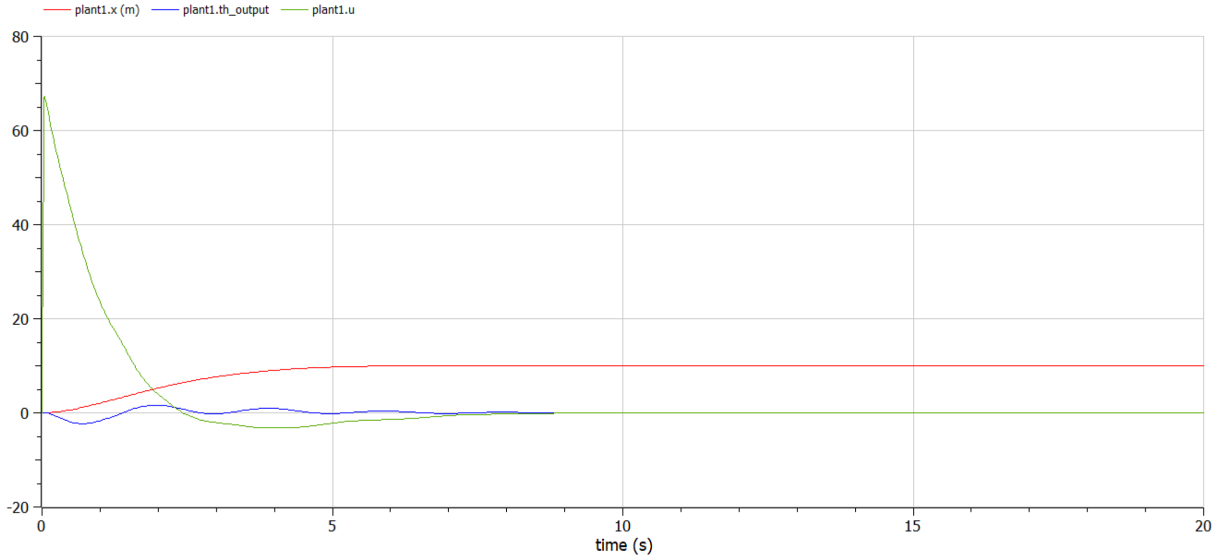

PID_Error
x(t)
Negate
Product
Error
Setpoint


PID_I
Error
Integral
Product
Result_I
I_init
Ki


PID_P
Error
Product
Result_P
Kp


PID_D
Error
Derivator
Product
Result_D
D_init
Kd


x
th
u
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35
40


plant1.x (m)
plant1.th_output
plant1.u
80
60
40
20
0
-20
0
5
10
15
20
time (s)