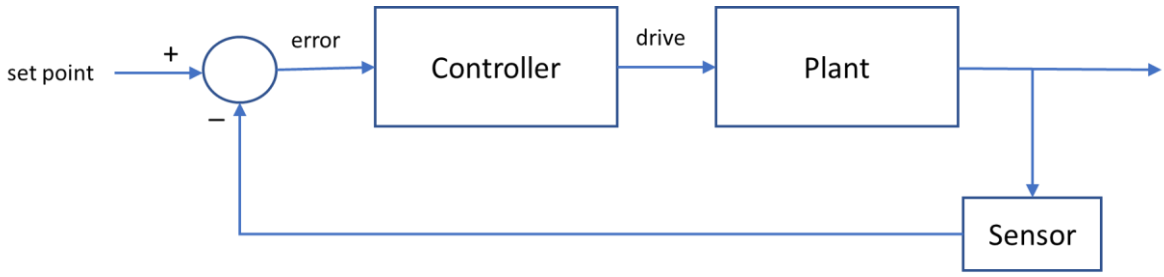
Comp3252 Modelling and Simulation. PID Control

CBPrice February 2021

The PID Controller

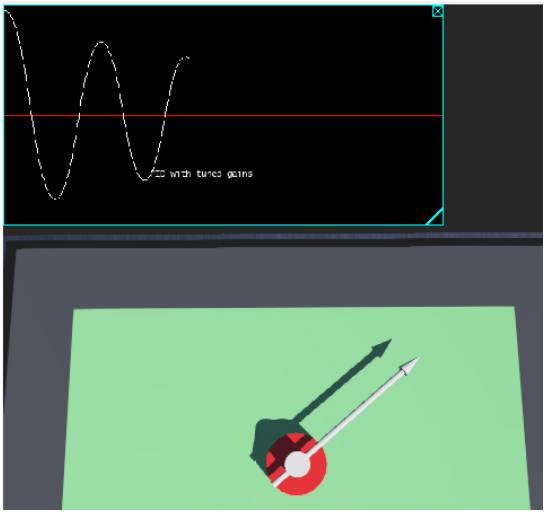
There's two very important software mini-architectures; Finite State Machines and Proportional-Integral-Derivative controllers. Interestingly both these mini-architectures have direct hardware implementations as well as software implementations. PID control is found in many engineering control systems from home-heating, car cruise control and aircraft control systems.



The user defines a set point, e.g. the desired cruise speed of a car. The plant consists of the engine and drive-chain and the sensor measures the car's road speed. The difference between the set point speed and error is calculated and sent to the controller. The controller uses the error value to send a drive signal to the plant (here giving more gas or less gas to the engine) so the error in speed converges to zero.

1 Investigating a PID controller

Here you will work with the Webot world **CBP_RotMot_PID_1.wbt** and the controller **CBP_3352_PID_1.c** Here's the scenario. The pointer start off pointing exactly East, and its target is exactly North. The display at the top shows how the error **TARGET_POSITION – position** where both these are angles, changes with time.



The controller has 3 control parameters Kp (proportional) Ki (integral) and Kd (derivative). These are input as **controllerArgs** in the Scene Tree as shown above right. The order from top down is Kp, Ki, Kd

(a) Start off with Ki=0, Kd = 0; and increase Kp = 5, then 10, then 20, then 30. For each use the Octave script **Plot_PID** to get an accurate plot of position versus time. Perhaps snip each and paste into a log.

See how the following change for each Kp: (i) The number of oscillations, (ii) The approx. speed by which the curve rises.

(b) Now set Kp = 40 and choose Kd = 1, 2, 3, 4, 5. Note down the effect of increasing Kd.

2 Ziegler-Nichols autotuning the parameters

Here we are going to use the above world to 'auto-tune' the parameters according to the Ziegler Nichols algorithm.

(a) Set Ki = 0 and Kd = 0 and increase Kp from a starting value of Kp = 20 until you can count 10 half-cycles which are clearly above or below the zero line. Note down your value of Kp. This is the 'critical' value of Kp.

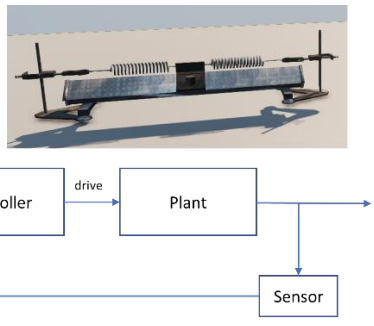
(b) Scroll through the console where the **Period** is logged. Estimate an average value of period for the first few half-cycles (ignore the -ve sign – this is fake). This is the critical time T.

(c) Run the Octave script **Z_N_Tuning** and input you K critical and T critical values. The script will output the 'tuned' gains.

(d) Set these gains into your Webots controller and record the behaviour of the system.

3 Coding the ODEs and Investigating the Model (UDK)

Here we shall look at a mass tethered by two springs. At time zero the mass is given a desired displacement of 1.0 and the controller makes this happen.



The error signal is coded as **error = desX - dispX;** where desX is the desired displacement and dispX is the actual displacement which varies with time

(a) Download the additional UDK assets and install these as usual. Accept any file replacements. The level you will run is **MK_PID_Test.udk** and the code for the actor is in the file **MAS14_MK_PID.uc**

(b) First we have to add to code for the plant (mass-spring system). You will recognize this code snippet

forceX = -stiffnessX*dispX - damping*velyX + u;
accelX = forceX/mmass;
 	
velyX += accelX*dT;
dispX += velyX*dT;
	Calculate the force from the spring and from the damping. Calculate the acceleration.
	
	Use the acceleration to update the velocity and therefore the displacement

(c) Now we have to code the controller as explained in the mini-lecture and the theory sheet

```
// save errors from previous time steps and get the latest error
e2 = e1;
e1 = error;
error = desX - dispX;

// Now calculate the PID controller
du = Kp*(error - e1) + (Ki*Ts)*error + (Kd/Ts)*(error - 2*e1 + e2);
u += du;
```

When you run the following investigations, point Octave to the UDK **Logs** folder and run the script **MK_PID_X** where X is the number of your latest experiment. The Octave file will plot your results and also a theoretical graph.

Remember **F1** starts the simulation and **F12** stops the simulation and flushes the Octave file to disk.

(d) Start with the following investigations. You can change the coefficients in the Editor (double left click on the actor to get its properties box)

Kp	Ki	Kd	Observe
50	0	0	Overshoot, oscillation, no convergence
25	25	0	Overshoot, oscillation, convergence
50	50	8	No overshoot, no oscillation, convergence

So you should have a good understanding what the three coefficients do

(e) Perform investigations to improve on the last result. Ideally you want (i) the displacement to correct itself as fast as possible, (ii) no overshoot or oscillations, (iii) good convergence.

Here's the table from the mini-lecture to remind you

Gain	**Rise Time**	**Overshoot**	**Settling Time**	**final error**
Kp	decrease	increase	small change	decrease
Ki	decrease	increase	increase	decrease
Kd	small change	decrease	decrease	no change