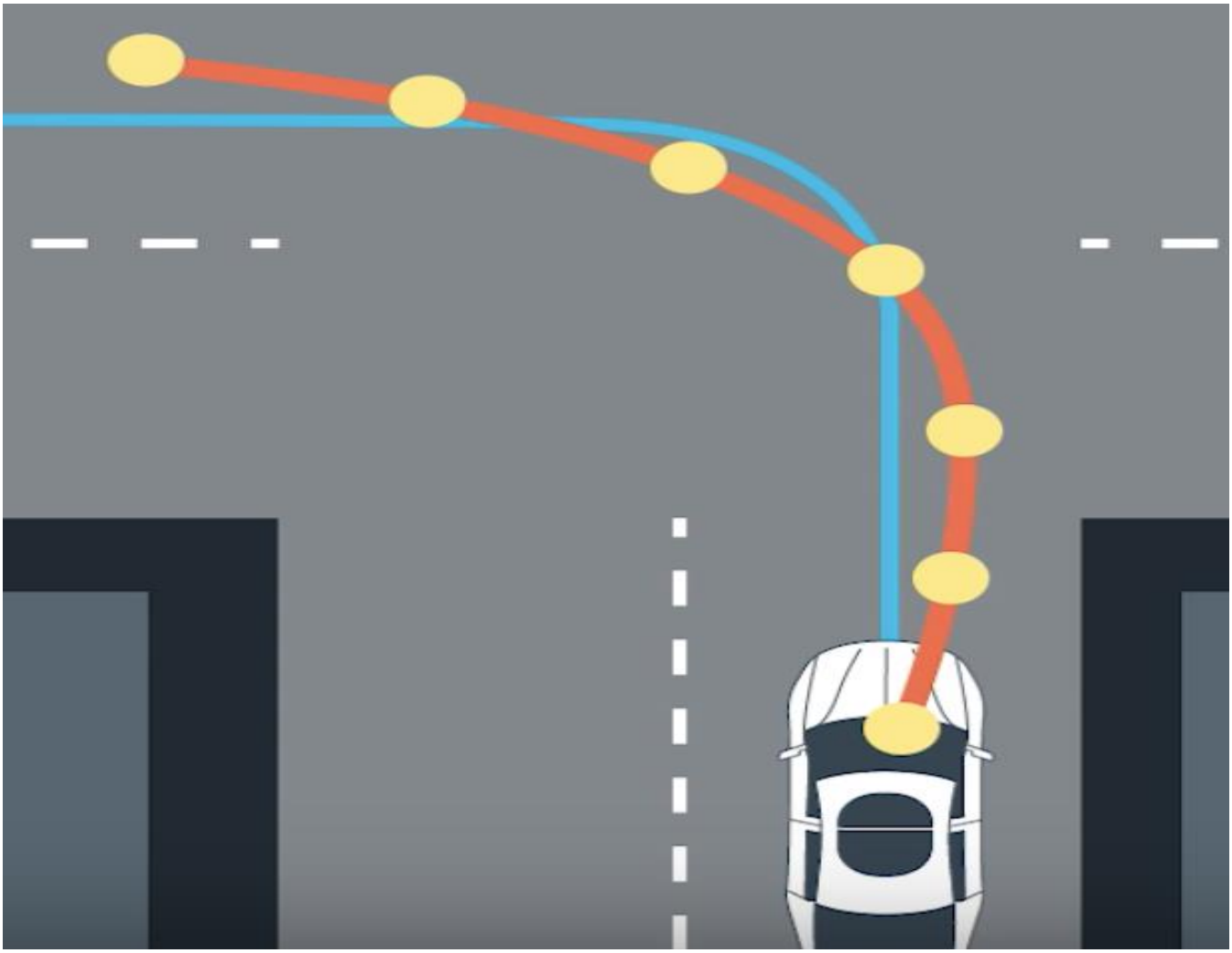
Control



Lab 3 : PID Controllers

Example 2: Cruise Control using PID controller:

The parameters for the system are as follows:

- (m) vehicle mass = 1000 kg
- (b) damping coefficient = 50 N.s/m

1. Model the Car cruise open loop system shown in Figure 9 and Equation (8) and get its open loop step response using attached MATLAB PID Simulator attached with the pdf. (if your MATLAB supports it), and by writing a MATLAB code.

- Plant transfer function G(S) = $\frac{V(s)}{U(s)} = \frac{1}{m\,S+\,b} = \frac{1}{1000\,S+\,50}$

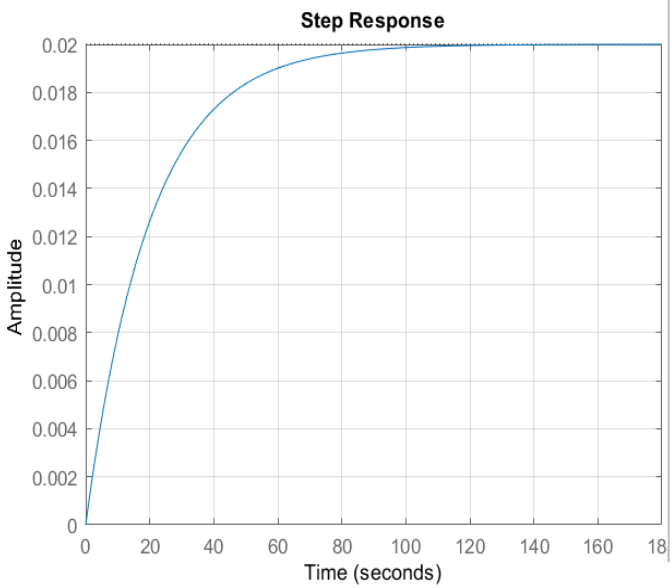
1) Open loop step response:

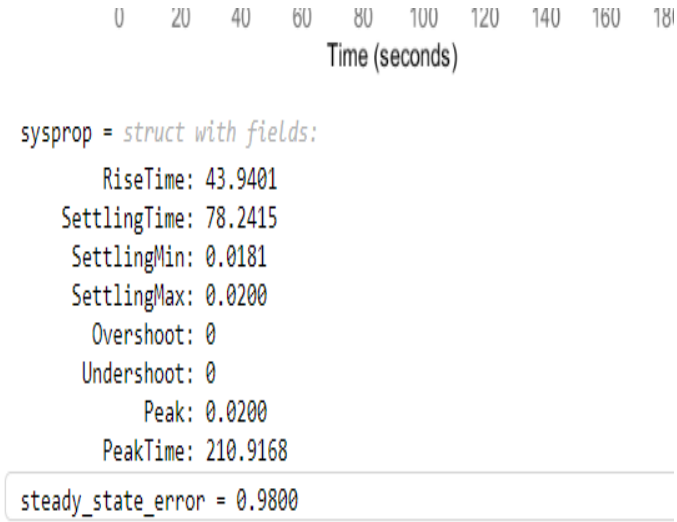
Code:

```
m = 1000;    % Vehicle mass (Kg)
b = 50;      % Damping coefficient (N.s/m)

S = tf("s");
plant_tf = 1/(m*S+ b)
[y ~] = step(plant_tf);
stepplot(plant_tf);
grid on;
% rlocusplot(plant_tf)
```

```
sysprop = stepinfo(plant_tf,'RiseTimeThreshold',[0.1 0.9]);
sysprop
steady_state_error = abs(1-y(end))
```

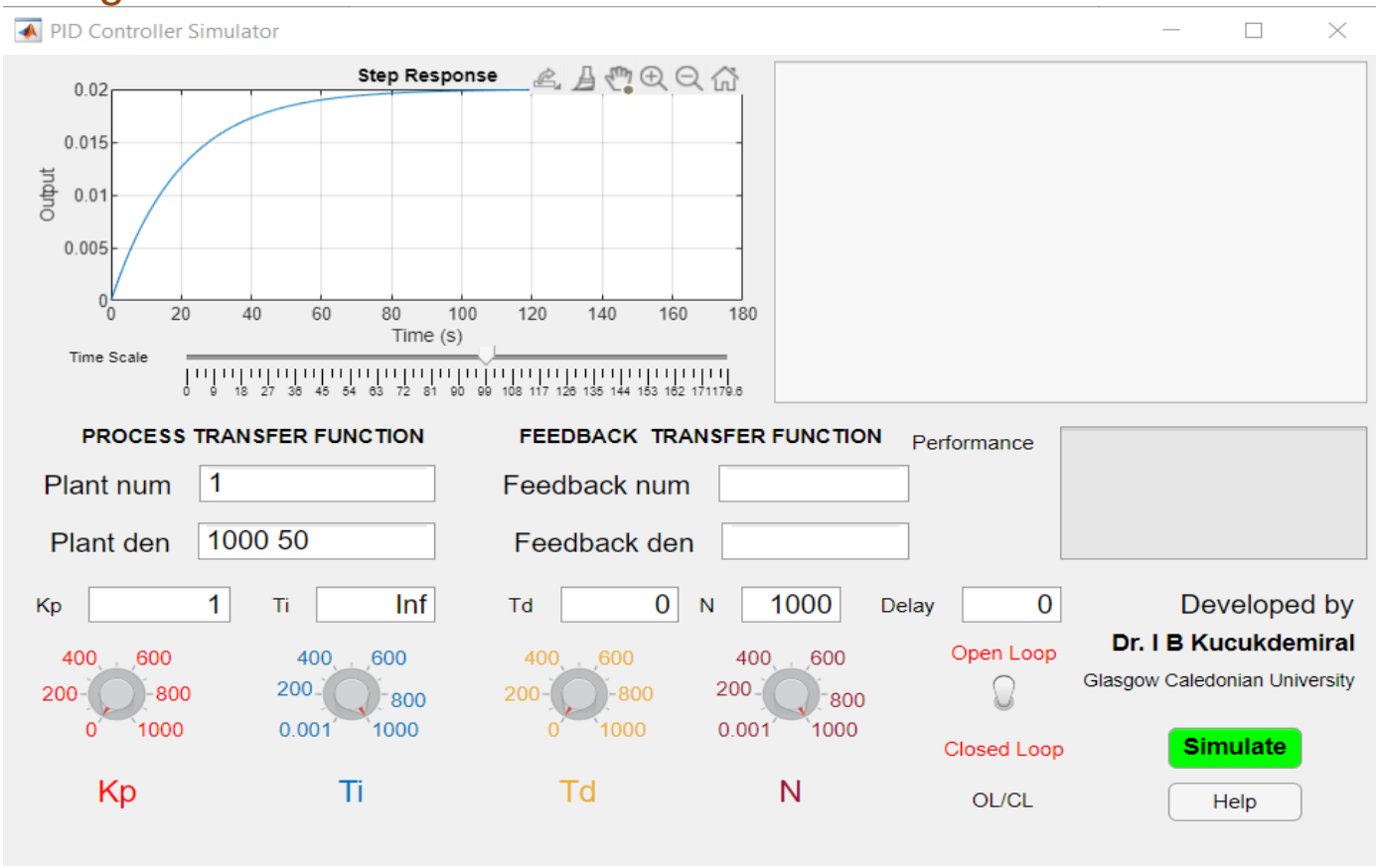





```
sysprop = struct with fields:
        RiseTime: 43.9401
    SettlingTime: 78.2415
     SettlingMin: 0.0181
     SettlingMax: 0.0200
       Overshoot: 0
      Undershoot: 0
            Peak: 0.0200
        PeakTime: 210.9168
```

```
steady_state_error = 0.9800
```

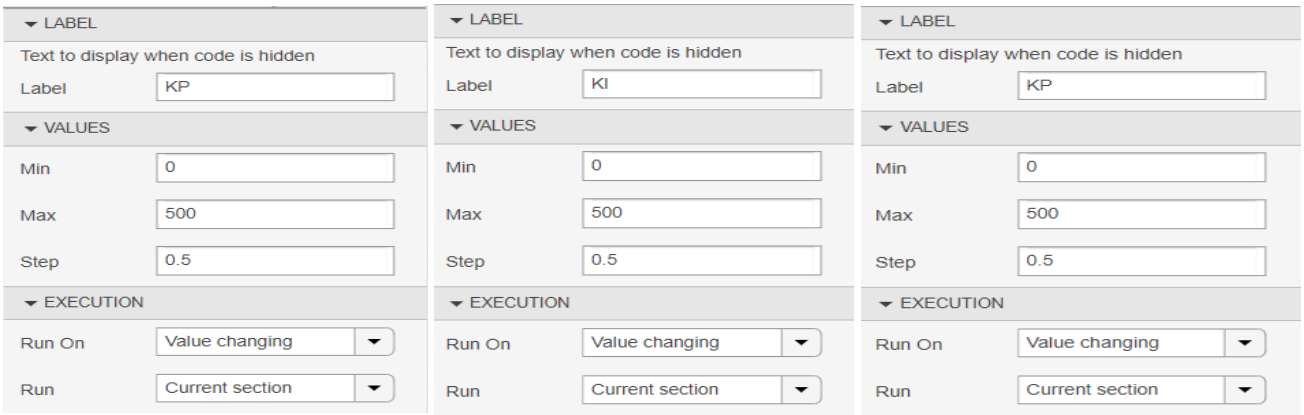
Using PID Simulator:



2. Adjust the PID parameters to get the following system specifications for a step response closed loop negative unity feedback system:

• Rise time < 5 sec • Overshoot < 10% • Steady-state error < 2%

I can't use the **Zeigler-Nichols** PID tuning method because there is only a pole at -0.05 so the root locus doesn't intersect the imaginary axes so I can't get ultimate gain (KU) and ultimate period (TU).
So I will use **Manual PID tuning** (trial and error):
I will use MATLAB live script and use the **numeric slider** for KP & KD & KI



1) I want to decrease **the rise time** from **43.9401 secs** in case of the open loop **to be less than 5 secs** so I need to add PID and increase **KP & KI.**

2) But increasing **KP & KI** increases the maximum overshoot so I need to increase **KD** which decrease the maximum overshoot.

3) Adding **KI** makes the steady state error to be almost zero because it increases the system type

I will choose:

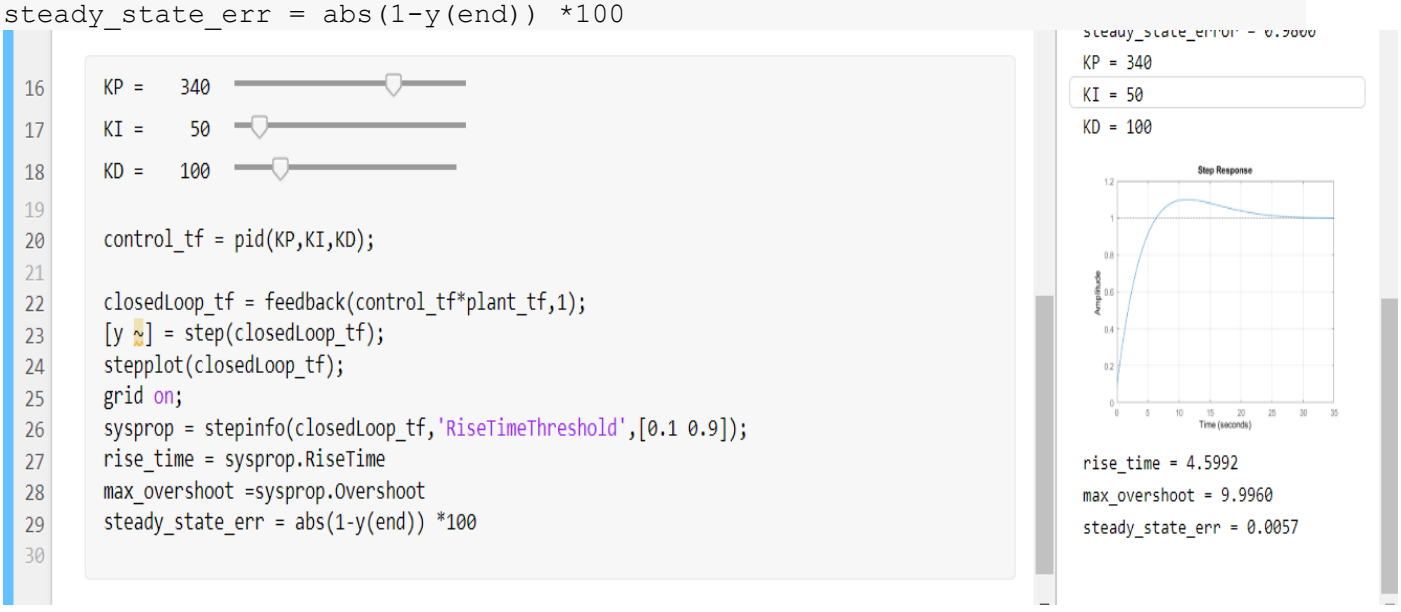
KP = 340 & KI = 50 & KD = 100

Code:

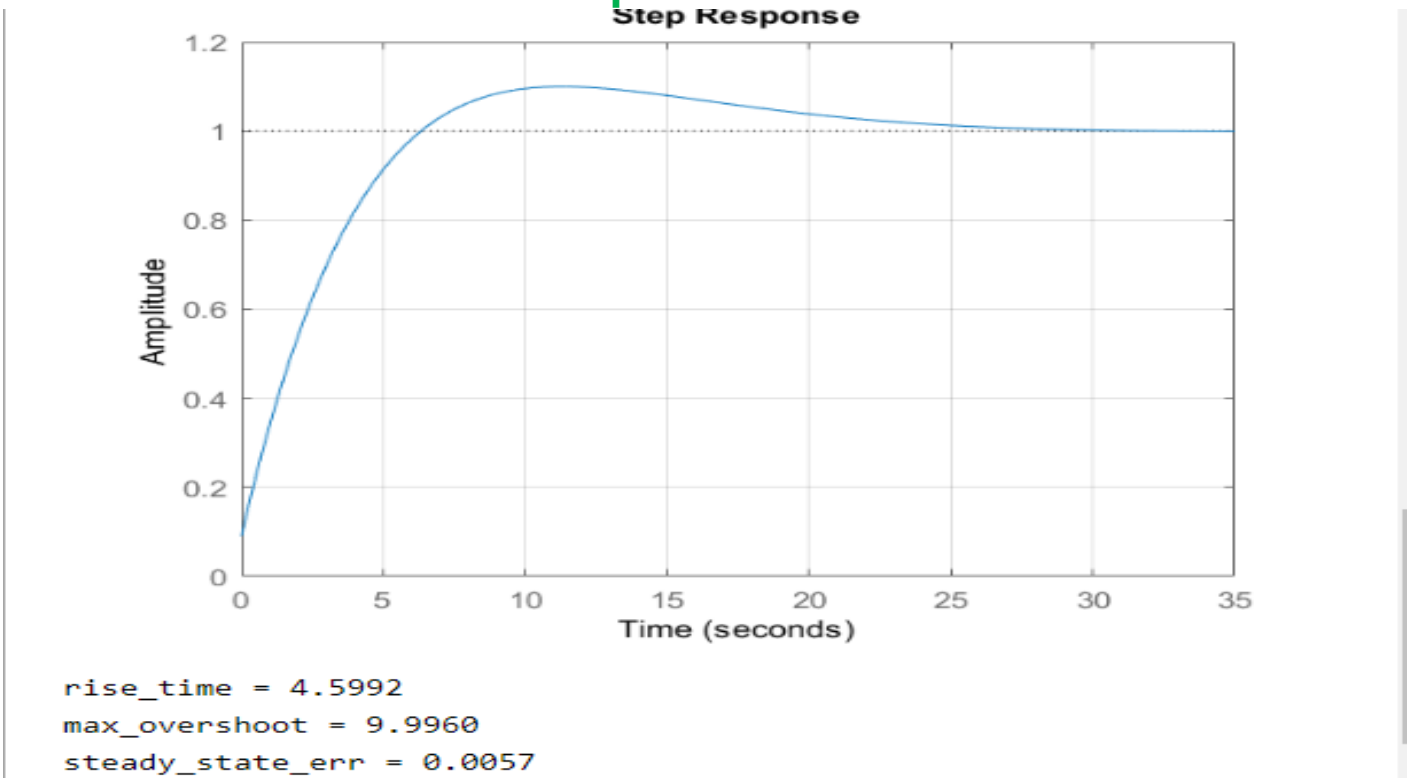
```
KP = 340
KI = 50
KD = 100


control_tf = pid(KP,KI,KD);


closedLoop_tf = feedback(control_tf*plant_tf,1);
[y ~] = step(closedLoop_tf);
stepplot(closedLoop_tf);
grid on;
sysprop = stepinfo(closedLoop_tf,'RiseTimeThreshold',[0.1 0.9]);
rise_time = sysprop.RiseTime
max_overshoot =sysprop.Overshoot
steady_state_err = abs(1-y(end)) *100
```



Proportional Integral Derivative Controller step response:




```
rise_time = 4.5992
max_overshoot = 9.9960
steady_state_err = 0.0057
```

Using PID Simulator: