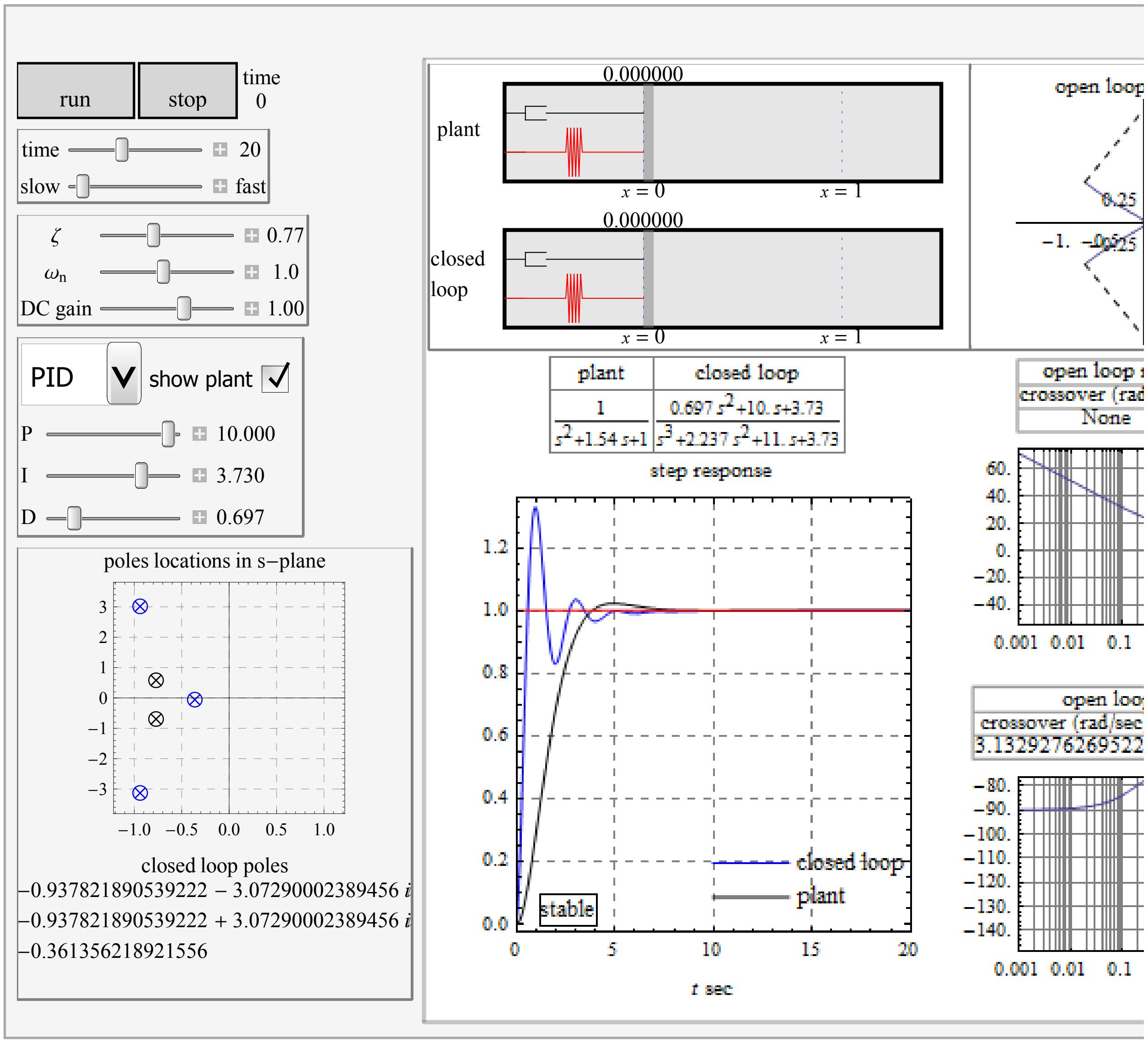
Simulation of Feedback Control System with Controller and Second-Order Plant

Initialization Code (optional)

Manipulate

```
Manipulate[
 tick;
 If[needToUpdateSystem,
  (
   needToUpdateSystem = False;

   (* generate the stepResponse once, only when the parameters change *)
   plant@updateStepResponse[s, t, 0, simulationTime];
   feedBack@updateStepResponse[s, t, 0, simulationTime];

   stepResponsePlot = feedBack@getPlot[s, t, simulationTime, showPlantResponse];
   bodePlot = feedBack@getBodePlot[];

   lowerGrid = Rasterize[(*rastersize for faster rendering*)
     Text@Grid[{
        {
         Grid[{{
             Text@Grid[{
                {Style["plant", 11], Style["closed loop", 11]},
                {TraditionalForm@plant@getPolynomial[s], TraditionalForm@feedBack@getPolynomial[s]}
               }, Spacings → {0, 0.3}, Frame → All, FrameStyle -> Directive[Thickness[.001], Gray]
              ]},
           {stepResponsePlot}
          }, Spacings → {0, 0}, Alignment → Center, Frame → None
         ],
         Grid[{
           {bodePlot[[1]]},
           {bodePlot[[2]]}
          }, Spacings → {0, 0}
         ]
        }
       }, Spacings → {0, 0}, Frame → None
      ]
    ];

   nyquistPlot =
    Rasterize@NyquistPlot[feedBack@getOpenLoopTF[s], ImageSize → {0.42 * ContentSizeW, 0.3 * ContentSizeH},
      AspectRatio → 1, ImagePadding → 0, ImageMargins → 0, PlotLabel -> Text@Style["open loop Nyquist", 11]];

   currentTime = 0
  )
 ];

 finalPlot = Grid[{
    {
     Grid[{
       {Grid[{{Text@Style["plant", 11], makeMechanicalSystem[plant@getStepResponse[] /. t → currentTime]},
            {Text@Style[Column[{"closed", "loop"}], 11],
             makeMechanicalSystem[feedBack@getStepResponse[] /. t -> currentTime]}}], nyquistPlot
          }
        }, Frame → All, FrameStyle -> Directive[Thickness[.005], Gray], Spacings → {0.2, 0}]
      },
      {lowerGrid, SpanFromLeft}
     }, Spacings → {0, 0}, Alignment → Center, Frame → None, FrameStyle -> Directive[Thickness[.005], Gray]
    ];

   If[doSimulation,
    (
     If[currentTime < simulationTime,
      (
       If[currentTime + delta ≥ simulationTime,
        (
         currentTime = simulationTime;
        ),
        (
         currentTime += delta
        )
       ];
       tick += $MachineEpsilon
      )
     ]
    )
   ];

   (*FinishDynamic[];*)
   finalPlot ,
   (*------ start controls ------------------*)
   Grid[{
     {Button[Text[Style["run", 12]],
       {
        currentTime = 0;
        doSimulation = True;
        tick += $MachineEpsilon
       }, ImageSize -> {60, 28}],
      Button[Text[Style["stop", 12]],
       {
        currentTime = 0;
        doSimulation = False;
        tick += $MachineEpsilon
       }, ImageSize -> {50, 28}],
      Dynamic[Text@Style[Grid[{{"time"}, {currentTime}}, Spacings → {0.1, 0.1}, Alignment → Center], 11]]
     }
    }, Spacings → {.2, 0}, Alignment → Center],

   Grid[{
     {Text@Style["time", 11],
      Manipulator[Dynamic[simulationTime, {
          simulationTime = #; needToUpdateSystem = True; currentTime = 0; tick += del} &],
       {1, 50, 1}, ImageSize → Tiny, ContinuousAction → False],
      Text@Style[Dynamic@padIt3[simulationTime, {2, 0}], 11]
     },
     {Text@Style["slow", 11],
      Manipulator[Dynamic[delta, {delta = #; currentTime = 0; tick += del} &],
       {0.1, 3, 0.1}, ImageSize → Tiny, ContinuousAction → False],
      Text@Style["fast", 11]
     }
    }, FrameStyle -> Directive[Thickness[.005], Gray], Frame → True, Spacings → {0.2, 0}],

   Grid[{
     {Text@Style["ζ", 11],
      Manipulator[Dynamic[zeta, {zeta = #; needToUpdateSystem = True; currentTime = 0; doSimulation = False;
```

```
        plant@setZeta[zeta]; tick += del} &], {0.01, 2.0, 0.01}, ImageSize → Tiny, ContinuousAction → False],
    Text@Style[Dynamic@padIt2[zeta, {3, 2}], 11]
   },

   {Text@Style["ωn", 11],
    Manipulator[Dynamic[wn,
      {wn = #; needToUpdateSystem = True; currentTime = 0; doSimulation = False; plant@setwn[wn]; tick += del} &],
     {0.1, 2., 0.1}, ImageSize → Tiny, ContinuousAction → False],
    Text@Style[Dynamic@padIt2[wn, {2, 1}], 11]
   },

   {Text@Style["DC gain", 11],
    Manipulator[Dynamic[dcGain, {dcGain = #; needToUpdateSystem = True;
        currentTime = 0; doSimulation = False; plant@setGain[dcGain]; tick += del} &],
     {0.01, 1.5, 0.01}, ImageSize → Tiny, ContinuousAction → False],
    Text@Style[Dynamic@padIt2[dcGain, {3, 2}], 11]
   }
  }, FrameStyle -> Directive[Thickness[.005], Gray], Frame → True, Spacings → {0.2, 0}],
 Grid[{
   {
    Row[{PopupMenu[Dynamic[controllerType, {controllerType = #; needToUpdateSystem = True;
          controller@setType[controllerType]; currentTime = 0; doSimulation = False; tick += del} &],
       {
         "P" → Style["P", 11],
        "PI" → Style["PI", 11],
        "PD" → Style["PD", 11],
        "PID" → Style["PID", 11]
       },
       ImageSize -> All
      ],
      Row[{Style[" show plant ", 11], Checkbox[
          Dynamic[showPlantResponse, {showPlantResponse = #; needToUpdateSystem = True; tick += del} &]]}]
     }], SpanFromLeft
   },
   {Text@Style["P", 11],
    Manipulator[Dynamic[propertional, {propertional = #; needToUpdateSystem = True;
        doSimulation = False; controller@setKp[propertional]; currentTime = 0; tick += del} &],
     {0.001, 10, 0.001}, ImageSize → Tiny, ContinuousAction → False
    ],
    Text@Style[Dynamic@padIt2[propertional, {5, 3}], 11]
   },
   {Text@Style["I", 11],
    Manipulator[Dynamic[integral, {integral = #; needToUpdateSystem = True; doSimulation = False;
        controller@setKi[integral]; currentTime = 0; tick += del} &], {0.0, 5, 0.001}, ImageSize → Tiny,
     ContinuousAction → False, Enabled → Dynamic[controllerType == "PI" || controllerType == "PID"]
    ],
    Text@Style[Dynamic@padIt2[integral, {4, 3}], 11]
   },
   {Text@Style["D", 11],
    Manipulator[Dynamic[derivative, {derivative = #; needToUpdateSystem = True; doSimulation = False;
        controller@setKd[derivative]; currentTime = 0; tick += del} &], {0.0, 5, 0.001}, ImageSize → Tiny,
     ContinuousAction → False, Enabled → Dynamic[controllerType == "PD" || controllerType == "PID"]
    ],
    Text@Style[Dynamic@padIt2[derivative, {4, 3}], 11]
   }
  }, Alignment → Left, FrameStyle -> Directive[Thickness[.005], Gray], Frame → True, Spacings → {0.3, 0.2}
 ],
 Grid[{
   {Text@Style["poles locations in s-plane", 11]},
   {Dynamic[Show[
      polesPlot[feedBack@getPoles[], Blue],
      If[showPlantResponse, polesPlot[plant@getPoles[s], Black], {}]
      , ImageMargins → 0, ImagePadding → {{20, 5}, {15, 0}}, ImageMargins → 0, AspectRatio → 1
     ]
    ]},
   {Dynamic[feedBack@formatClosedLoopPoles[]]}
  }, Alignment → Center, FrameStyle -> Directive[Thickness[.002], Gray], Frame → True, Spacings → {0, 0}
 ],

 {{tick, 0}, None},
 {{del, $MachineEpsilon}, None},
 {{doSimulation, False}, None},
 {{currentTime, 0}, None},
 {{propertional, 10.0}, None},
 {{integral, 3.730}, None},
 {{derivative, 0.697}, None},
 {{finalPlot, {}}, None},
 {{controllerType, "PID"}, None},
 {{delta, .2}, None},
 {{needToUpdateSystem, True}, None},
 {{showPlantResponse, True}, None},
 {{simulationTime, 20}, None},
 {{zeta, 0.77}, None},
 {{wn, 1.0}, None},
 {{dcGain, 1.0}, None},
 {{lowerGrid, {}}, None},
 {{bodePlot, {}}, None},
 {{stepResponsePlot, {}}, None},
 {{nyquistPlot, {}}, None},
 {plant, None},
 {controller, None},
 {feedBack, None},
 TrackedSymbols :→ {tick},

 SynchronousUpdating → False,
 ContinuousAction → False,
 SynchronousInitialization → True,
 Alignment -> Center,
 ImageMargins → 0,
 FrameMargins → 0,
 Paneled → True,
 Frame → False,
 ControlPlacement → Left,
 Initialization :→
  {ContentSizeW = 410;
   ContentSizeH = 470 ;

   (*-----------------------------------------*)
   padIt2[v_ ? (NumericQ[#] && Im[#] == 0 &), f_List] :=
    AccountingForm[Chop[N@v] , f, NumberSigns -> {"", ""}, NumberPadding -> {"0", "0"}, SignPadding -> True];
   (*-----------------------------------------*)
   padIt3[v_ ? (NumericQ[#] && Im[#] == 0 &), f_List] := AccountingForm[Chop[N@v] , f,
     NumberSigns -> {"", ""}, NumberPadding -> {"0", "0"}, SignPadding -> True, NumberPoint → ""];
   (*-----------------------------------------*)
   (*Plant class*)
   plantClass[$s_, $t_, $from_, $to_, $zeta_, $wn_, $gain_] :=
    Module[{zeta = $zeta, wn = $wn, gain = $gain, stepResponse, self},

     self@getStepResponse[s_, t_, from_, to_] := (
       First@OutputResponse[self@getTF[s], UnitStep[t], {t, from, to}]
      );

     self@setZeta[v_] := (
       zeta = v;
      );

     self@setwn[v_] := (
        wn = v;
      );

     self@setGain[v_] := (
       gain = v;
      );

     self@getTF[s_] := (
       TransferFunctionModel[(wn^2*gain) / (s^2 + 2*zeta*wn*s + wn^2), s]
      );

     self@getPolynomial[s_] := (wn^2*gain) / (s^2 + 2*zeta*wn*s + wn^2);
     self@updateStepResponse[s_, t_, from_, to_] := stepResponse = self@getStepResponse[s, t, from, to];
     self@getStepResponse[] := stepResponse;

     self@getPoles[s_] := TransferFunctionPoles[self@getTF[s]];

     stepResponse = self@getStepResponse[$s, $t, $from, $to];
     self
    ];
  (*------------------------------------------*)
     (*controller class*)
   controllerClass[$type_, $kp_, $ki_, $kd_] := Module[{type = $type, kp = $kp, ki = $ki, kd = $kd, self},

     self@setType[v_] := type = v;
     self@setKp[v_] := kp = v;
     self@setKi[v_] := ki = v;
     self@setKd[v_] := kd = v;

     self@getPolynomial[s_] := (
       Which[type == "PID", kp + ki / s + kd*s,
        type == "PD", kp + kd*s,
        type == "PI", kp + ki / s,
        type == "P", kp
       ]
      );

     self
    ];
   (*------------------------------------------*)
     (*closed loop feedback class*)
   feedBackClass[$s_, $t_, $from_, $to_] := Module[{getOpenLoopPolynomial, stepResponse, self},

     getOpenLoopPolynomial[s_] := controller@getPolynomial[s]*plant@getPolynomial[s];
     self@getStepResponse[s_, t_, from_, to_] :=
      First@OutputResponse[ self@getTF[s], UnitStep[t], {t, from, to}];

     self@getOpenLoopTF[s_] := TransferFunctionModel[getOpenLoopPolynomial[s], s];
     self@getOpenLoopPhaseMargins[s_] := PhaseMargins[self@getOpenLoopTF[s]];
     self@getOpenLoopGainMargins[s_] := GainMargins[self@getOpenLoopTF[s]];

     self@getPolynomial[s_] := Module[{p = getOpenLoopPolynomial[s]},
       N@Simplify[p / (1 + p)]
      ];

     self@getTF[s_] := TransferFunctionModel[self@getPolynomial[s], s];

     self@updateStepResponse[s_, t_, from_, to_] := stepResponse = self@getStepResponse[s, t, from, to];
     self@getStepResponse[] := stepResponse;

     self@getPoles[] := Module[{s},
       TransferFunctionPoles[self@getTF[s]]
      ];
     self@isStable[] := Module[{poles = self@getPoles[]},
       poles = Flatten[Re[poles]];
       If[Length[Cases[poles, n_ /; n > 0, {}]] > 0, False, True]
      ];

     (**-----------------------------------------**)
     self@getPlot[s_, t_, simulationTime_, showPlantResponse_] := Module[{plotOptions, line, plantPlot,
        closedLoopPlot, plantPlotStyle = {Black, Thin}, closedLoopPlotStyle = {Blue, Thin}},

       plotOptions = {AxesOrigin → {0, 0}, PlotRange → {{0, simulationTime}, All},
         ImagePadding → {{30, 15}, {40, 20}}, Frame → True, ImageSize →
          {0.65 * ContentSizeW, 0.6 * ContentSizeH}, AspectRatio → 1.1, ImageMargins → 0, GridLines → Automatic,
         GridLinesStyle -> Directive[Thickness[.001], Gray, Dashed], Axes → False, Exclusions → None};

       line = ListPlot[{{0, 1}, {simulationTime, 1}}, PlotStyle → Red, Joined → True];

       plantPlot = Plot[plant@getStepResponse[],
         {t, 0, simulationTime}, Evaluate@plotOptions, PlotStyle → plantPlotStyle];

       closedLoopPlot = Plot[self@getStepResponse[], {t, 0, simulationTime},
         Evaluate@plotOptions, PlotStyle → closedLoopPlotStyle,
         FrameLabel → {{None, None},
           {Text@Style[Row[{Style["t", Italic], " sec"}], 11], Text@Style["step response", 11]}},

         Epilog → {

           If[self@isStable[],
             Text[Framed[Style["stable", Black, 12], FrameMargins → .1], Scaled[{.06, .05}], {-1, 0}],
             Text[Framed[Style["unstable", Red, 12], FrameMargins → .1], Scaled[{.06, .05}], {-1, 0}]
           ],

           makePlotLegend[{"closed loop", If[showPlantResponse, "plant", ""]},
              (Graphics[{Directive[#], Line[{{-4, 0}, {4, 0}}]}]) & /@
               {closedLoopPlotStyle, If[showPlantResponse, plantPlotStyle, White]},
              {0.56, 0.2}, 14, 12, "Arial"]

          }
        ];

       Show[closedLoopPlot, If[showPlantResponse, plantPlot, {}], line, ImageMargins → 0]
      ];
     (**-----------------------------------------**)
     self@getBodePlot[] := Module[{bodePlotTitle, phasePlotTitle, pm, gm, s},

       pm = Last@Sort@self@getOpenLoopPhaseMargins[s];
       gm = Last@Sort@self@getOpenLoopGainMargins[s];

       bodePlotTitle = Text@Style[
          Grid[{
              {"open loop magnitude plot", SpanFromLeft},
              {"crossover (rad/sec)", "margin (db)"},
              {gm[[1]], 20 * Log[10, gm[[2]]]}
           }, Frame → All, FrameStyle -> Directive[Thickness[.001], Gray],
           Spacings → {0.2, 0.15}, Alignment → Center], 11];

       phasePlotTitle = Text@Style[
          Grid[{
              {"open loop phase plot", SpanFromLeft},
              {"crossover (rad/sec)", "margin (deg)"},
              {pm[[1]], 180 / Pi pm[[2]]}
           }, Frame → All, FrameStyle -> Directive[Thickness[.001], Gray],
           Spacings → {0.2, 0.15}, Alignment → Center], 11];

       BodePlot[
        self@getOpenLoopTF[s],
        StabilityMargins → {True, True},
        StabilityMarginsStyle → {Red, Red},
        GridLines → Automatic,
        ImagePadding → {{{30, 5}, {30, 5}}, {{30, 5}, {30, 5}}},
        ImageMargins → {0, 0.1},
        PlotLabel → {bodePlotTitle, phasePlotTitle},
        PlotLayout → "List"
       ]
      ];

     (**----------------------------------------**)
     self@formatClosedLoopPoles[] := Module[{tbl = Table["", {4}], poles = Flatten@self@getPoles[]},
       tbl[[1 ;; Length[poles]]] = poles;
       Text@Style[Grid[{
           {Text@Style["closed loop poles", 11]},
           {Column[tbl]}}, Frame → None, Spacings → {0, 0.2}, Alignment → Center], 11]
      ];

     stepResponse = self@getStepResponse[$s, $t, $from, $to];
     self
    ];

   (*-----------------------------------------*)
   polesPlot[poles_, plotStyle_] := Module[{polesPoints, plotOptions},

     plotOptions = {AxesOrigin → {0, 0}, ImagePadding → {{0, 0}, {0, 5}}, Frame → True, ImageMargins → 0,
       AspectRatio → 1, GridLines → Automatic, GridLinesStyle -> Directive[Thickness[.001], Gray, Dashed]};
     polesPoints = makePolesAndZerosCoordinates[poles];

     Show[ListPlot[polesPoints,
       AxesOrigin → {0, 0},
       PlotMarkers → Style["⊗", plotStyle, 12],
       Evaluate@plotOptions
      ],
      PlotRange → {
         { Min[-1.2, 1.2 * Min[polesPoints[[All, 1]] ] ], Max[1.2, 1.2 * Max[polesPoints[[All, 1]]] ]},
         { Min[-1.2, 1.2 * Min[polesPoints[[All, 2]] ] ], Max[1.2, 1.2 * Max[polesPoints[[All, 2]]] ]}
       },
      ImageSize → {.35 * ContentSizeW, .3 * ContentSizeH},
      ImageMargins → 0,
      ImagePadding → 0
     ]
    ];
   (*-----------------------------------------*)
   makePolesAndZerosCoordinates[points_] := Module[{xy},
     xy = Flatten[Replace[points, {Complex[x_, y_] :→ {x, y}, x_?NumericQ :→ {x, 0}}, {3}], 1];
     Cases[xy, {_?NumericQ, _?NumericQ}, {2}]
    ];
   (*-----------------------------------------*)
   makePlotLegend[names_, markers_, origin_, markerSize_, fontSize_, font_] := Module[{i, idx},
     idx = Flatten[Position[names, _? (Not[# == ""] &)]];

     Join @@ Table[
       {
        Text[
         Style[names[[idx[[i]]]], FontSize → fontSize, font],
         Offset[{1.5 * 1.5 * markerSize, -(i - 0.5) * Max[markerSize, fontSize] * 1.25}, Scaled[origin]],
         {-1, 0}
        ],
        Inset[
         Show[
          markers[[idx[[i]]]],
          ImageSize → 3 * markerSize
         ],
         Offset[
           {markerSize / 2, -(i - 0.5) * Max[markerSize, fontSize] * 1.25},
           Scaled[origin]
         ],
         {0, 0},
         Background → Directive[Opacity[0], White]
        ]
       }
       ,
        {i, 1, Length[idx]}
       ]
     ];
     (*----------------------------------------*)
     (*Thanks to Árpád Kósa for this
      function below which draws the spring. From a demo found at WRI site*)
   rugo[xkezd_, ykezd_, xveg_, yveg_] := Module[{step = 12, szel = 0.25
        (*spring width*), hx, hy, veghossz = 0.3, hossz, dh, i},
       {
       hx = xveg - xkezd;
       hy = yveg - ykezd;
       hossz = Sqrt[hx^2 + hy^2] ;
       dh = (hossz - 2 * veghossz) / step;
        {xkezd, ykezd}} ~ Join ~ {{xkezd + hx * (dh + veghossz) / hossz, ykezd + hy * (dh + veghossz) / hossz}} ~
      Join ~ Table[If[OddQ[i], {xkezd + hx * (i * dh + veghossz) / hossz + hy * szel / hossz,
         ykezd + hy * (i * dh + veghossz) / hossz - hx * szel / hossz},
         {xkezd + hx * (i * dh + veghossz) / hossz - hy * szel / hossz,
         ykezd + hy * (i * dh + veghossz) / hossz + hx * szel / hossz}], {i, 2, (step - 2)}] ~ Join ~
        {{xkezd + hx * ((step - 1) * dh + veghossz) / hossz, ykezd + hy * ((step - 1) * dh + veghossz) / hossz}} ~
      Join ~ {{xveg, yveg}}
   ];
(*-----------------------------------------*)
   makeMechanicalSystem[y$_] := Module[{a1 = 1, a2, a3 = .7, a4, a6 = .3, annot, a0 = .7, y = y$},
     a2 = .15 * a0;
     a4 = .3 * a0;
       (*add virtual stop guards for unstable system*)
     If[y > 1.5, y = 1.5];
     If[y < -0.2, y = -0.2];

     annot =
      Row[{PaddedForm[(Chop[N@y, 10^-5]), {6, 6}, NumberSigns → {"", ""}, NumberPadding → {"0", "0"}]}];

     Graphics[{
        {EdgeForm[Directive[Thick, Black]], GrayLevel[0.9], Rectangle[{0, 0}, {1.5 + a0, a1}]},
        {GrayLevel[0.5], Opacity[.5], Rectangle[{y + a0, 0}, {y + a0 + 0.05, a1}]},
        {Black, Line[{{a4, a3}, {y + a0, a3}}] }, (*damper to mass line*)
        {Black, Line[{{0, a3}, {a2, a3}}] }, (*bottom of damper to left wall*)
        {Black, Line[{{a2, .9 a3}, {a2, 1.1 a3}}] }, (*bottom of damper*)
        {Black, Line[{{a2, .9 a3}, {a4, .9 a3}}] }, (* bottom edge of damper dash*)
        {Black, Line[{{a2, 1.1 a3}, {a4, 1.1 a3}}] }, (* top edge of damper dash*)
        {Black, Line[{{0, a6}, {a2, a6}}] }, (*line from left wall to start of spring part*)
        {Blue, Thin, Dotted, Line[{ {a0 + 1, 0}, {a0 + 1, a1}}] }, (*step response reference*)
        {Blue, Thin, Dotted, Line[{{a0, 0}, {a0, a1}}]}, (*0 initial position line*)
        {Text[Style[annot, 11], {y + a0, 1.1 * a1}, {0, 0}]  },
        {Text[Row[{Style["x", Italic], Style[" = 0", 11]}], {a0, -0.1}, {0, 0}] },
        {Text[Row[{Style["x", Italic], Style[" = 1", 11]}], {a0 + 1, -0.1}, {0, 0}] },
        {Red, Line[rugo[0, a6, y + a0, a6]]}
       }, PlotRange → {{-0.01, 2.3}, {-.21, 1.2 a1}}, Axes → False, AxesOrigin → {0, 0}, ImageMargins → 0,
       ImagePadding → 0, ImageSize → {0.58 * ContentSizeW, 0.15 * ContentSizeH}, AspectRatio → .3]
     ];

    (*make plant, controller and feedback objects*)
    plant = plantClass[s, t, 0, 20, .77, 1, 1];
    controller = controllerClass["PID", 10.0, 3.730, 0.697];
    feedBack = feedBackClass[s, t, 0, 20];
   }

]
```



ζ 0.77
ω_n 1.0
DC gain 1.00

PID show plant ✓

P 10.000
I 3.730
D 0.697

plant	closed loop
$\frac{1}{s^2+1.54\,s+1}$	$\frac{0.697\,s^2+10.\,s+3.73}{s^3+2.237\,s^2+11.\,s+3.73}$

open loop
crossover (rad
None

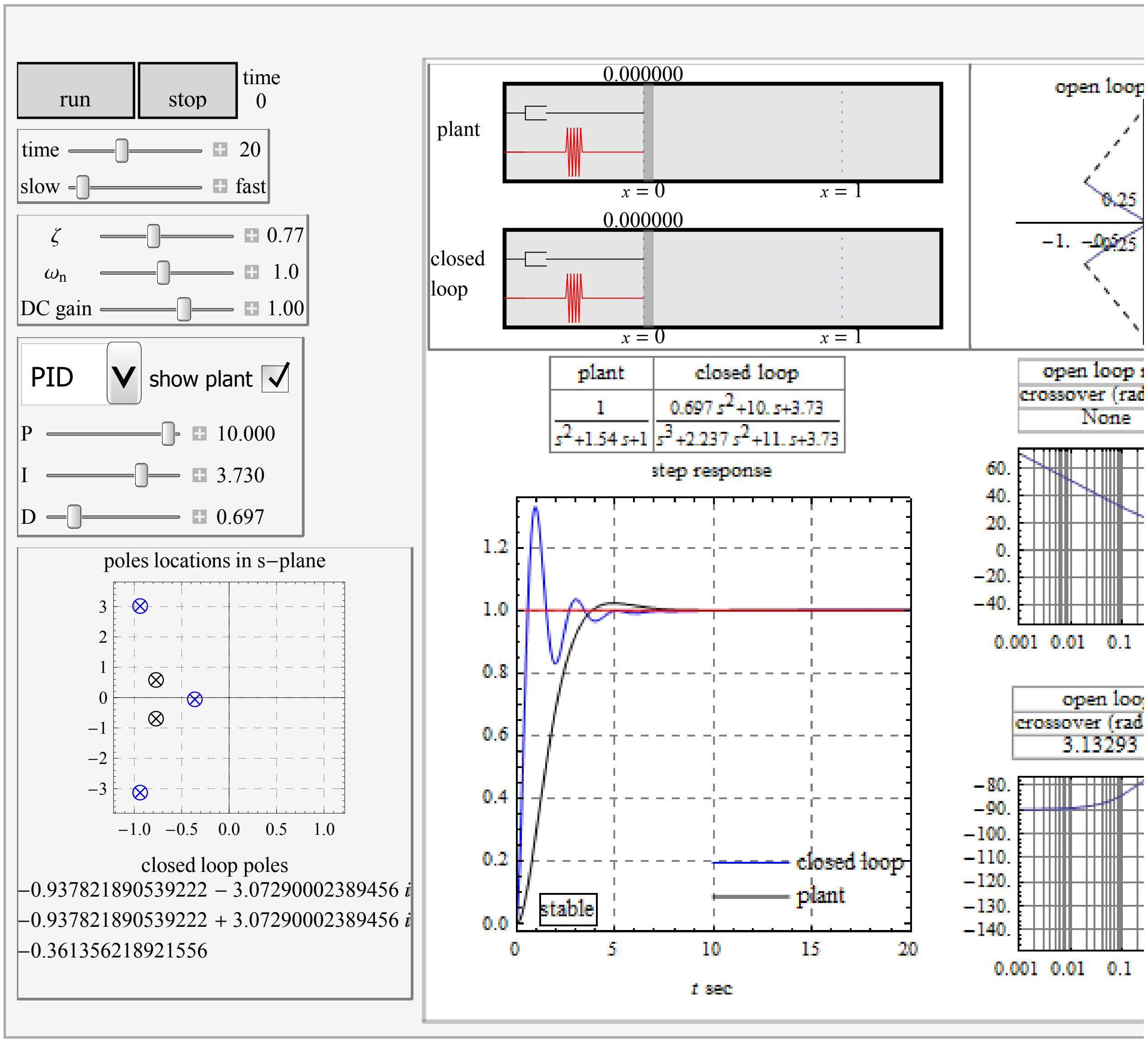
open loo
crossover (rad/sec
3.1329276269522

closed loop poles
−0.937821890539222 − 3.07290002389456 i
−0.937821890539222 + 3.07290002389456 i
−0.361356218921556

Caption

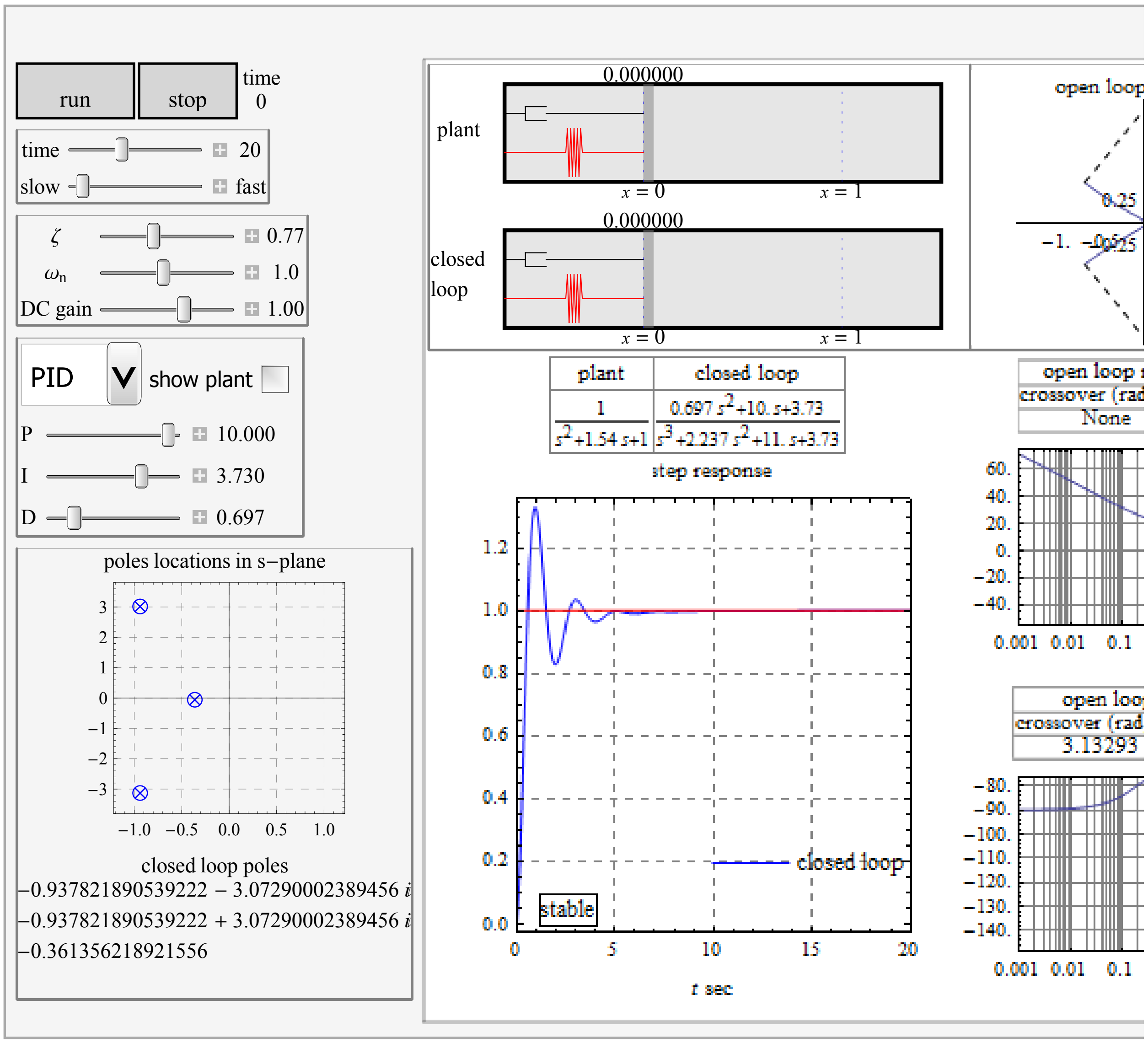
This Demonstration simulates a second-order linear system represented as a mass-spring-damper and analyzes the effect of adding a controller on the step response of the overall system in a closed loop. You can select one of four types of controller: P, PI, PD, or PID. You can simulate the plant with and without the controller to see the effect on the step response. Bode and Nyquist plots of the open-loop transfer functions are plotted and the gain and phase margins are shown, as well as the locations of the closed-loop poles. You can change the simulation time and speed as well as the controller parameters using sliders.

Thumbnail



plant	closed loop
$\frac{1}{s^2+1.54\,s+1}$	$\frac{0.697\,s^2+10.\,s+3.73}{s^3+2.237\,s^2+11.\,s+3.73}$

closed loop poles
−0.937821890539222 − 3.07290002389456 *i*
−0.937821890539222 + 3.07290002389456 *i*
−0.361356218921556



time 0

time 20

slow fast

ζ 0.77

ω_n 1.0

DC gain 1.00

PID show plant

P 10.000

I 3.730

D 0.697

closed loop poles
–0.937821890539222 – 3.07290002389456 *i*
–0.937821890539222 + 3.07290002389456 *i*
–0.361356218921556

plant	closed loop
$\frac{1}{s^2+1.54\,s+1}$	$\frac{0.697\,s^2+10.\,s+3.73}{s^3+2.237\,s^2+11.\,s+3.73}$

open loop
crossover (rad
None

open loo
crossover (rad
3.13293

Details (optional)

A feedback control system is simulated using a controller and a second-order plant. The controller operation can be proportional, proportional-integral, proportional-derivative, or proportional-integral-derivative. *Mathematica* 8 control system functions are used in all the computation.
To determine relative stability of the closed loop, the gain margin in db and phase margin in degrees of the open loop are shown above the Bode plots. For the case of multiple crossover frequencies, the highest frequency is used. The plant and closed-loop transfer functions are shown above the step response plot. For analysis of the absolute stability, the closed-loop poles' locations on the s-plane as well as their coordinates are displayed below the plot in the left side. The closed loop is unstable when the real part of any pole is positive. SI units are used throughout.

The plant transfer function is given by $G(s)=\frac{Y(s)}{U(s)}=\frac{\mathrm{dc}\,\omega_n^2}{s^2+2\,\zeta\,\omega_n\,s+\omega_n^2}$, where dc is the DC gain, ζ is the damping ratio, ω_n is the natural frequency, $Y(s)$ is the Laplace transform of the output of the plant, and $U(s)$ is the Laplace transform of the input to the plant. The controller used in the parallel form is given by $C(s)=k_p+k_d\,s+k_i/s$, where k_p, k_d, and k_i are the proportional, derivative, and integral parameters, respectively. The open-loop transfer function is $L(s)=C(s)\,G(s)$ and the closed-loop transfer function is $\frac{L(s)}{1+L(s)}$.

References

[1] J. J. D'Azzo and C. H. Houpis, *Linear Control System Analysis & Design*, 3rd ed., New York: McGraw–Hill, 1988.

[2] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 11th ed., Englewood Cliffs, NJ: Prentice–Hall, 2007.

[3] K. Ogata, *System Dynamics*, 4th ed., Englewood Cliffs, NJ: Prentice–Hall, 2003.

Control Suggestions (optional)

- ✓ Resize Images
- ☐ Rotate and Zoom in 3D
- ☐ Drag Locators
- ☐ Create and Delete Locators
- ✓ Slider Zoom
- ☐ Gamepad Controls
- ☐ Automatic Animation
- ☐ Bookmark Animation

Search Terms (optional)

PID
controller
second order system
mass-spring-damper
Bode
Nyquist
closed loop
transfer function

Related Links (optional)

Proportional-Integral-Derivative Method
Control Systems
Control Systems

Authoring Information

Contributed by: Nasser M. Abbasi