5ELEN018W - Robotic Principles
Lecture 8: Control - Part 3

Dr Dimitris C. Dracopoulos

The Laplace Transform

The Laplace Transform

Linear differential equations describing physical dynamic systems (including robots) can be transformed to algebraic equations which can be more easily solved (and also analyse) using the Laplace transform.

The Laplace Transform

Linear differential equations describing physical dynamic systems (including robots) can be transformed to algebraic equations which can be more easily solved (and also analyse) using the Laplace transform.

- The robotic arm performing a surgery (mass spring damper example), seen last week:

The Laplace Transform

Linear differential equations describing physical dynamic systems (including robots) can be transformed to algebraic equations which can be more easily solved (and also analyse) using the Laplace transform.

- The robotic arm performing a surgery (mass spring damper example), seen last week:

$$m\ddot{x} + b\dot{x} + kx = f \tag{1}$$

The Laplace Transform

Linear differential equations describing physical dynamic systems (including robots) can be transformed to algebraic equations which can be more easily solved (and also analyse) using the Laplace transform.

- The robotic arm performing a surgery (mass spring damper example), seen last week:

$$m\ddot{x} + b\dot{x} + kx = f \tag{1}$$

assuming all initial conditions are set to 0, then its Laplace transform is:

The Laplace Transform

Linear differential equations describing physical dynamic systems (including robots) can be transformed to algebraic equations which can be more easily solved (and also analyse) using the Laplace transform.

- The robotic arm performing a surgery (mass spring damper example), seen last week:

$$m\ddot{x} + b\dot{x} + kx = f \tag{1}$$

assuming all initial conditions are set to 0, then its Laplace transform is:

$$ms^2 + bs + k = F \tag{2}$$

Transfer Functions

Transfer Functions

The transfer function of a *linear, time-invariant* system is defined as the ratio of the Laplace transform of the output variable $Y(s)$ to the Laplace transform of the input variable $R(s)$,

Transfer Functions

The transfer function of a *linear, time-invariant* system is defined as the ratio of the Laplace transform of the output variable $Y(s)$ to the Laplace transform of the input variable $R(s)$, with all initial conditions assumed to be 0:

Transfer Functions

The transfer function of a *linear, time-invariant* system is defined as the ratio of the Laplace transform of the output variable $Y(s)$ to the Laplace transform of the input variable $R(s)$, with all initial conditions assumed to be 0:

$$G(s) = \frac{Y(s)}{R(s)} \tag{3}$$

Transfer Functions

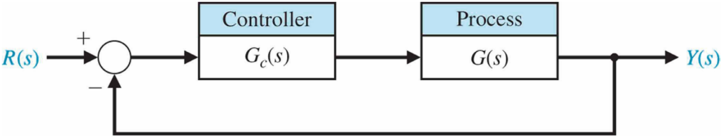
The transfer function of a *linear, time-invariant* system is defined as the ratio of the Laplace transform of the output variable $Y(s)$ to the Laplace transform of the input variable $R(s)$, with all initial conditions assumed to be 0:

$$G(s) = \frac{Y(s)}{R(s)} \tag{3}$$

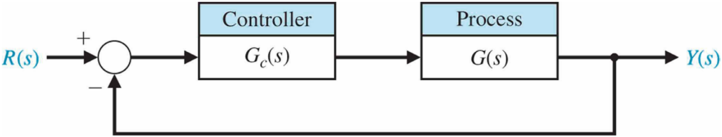
- These are used to make easier the modelling and analysis of dynamic systems.

Feedback (Closed-Loop) Controllers Transfer Function

Feedback (Closed-Loop) Controllers Transfer Function



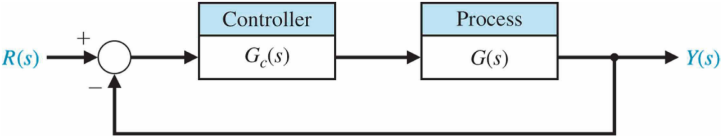
Feedback (Closed-Loop) Controllers Transfer Function



The transfer function of the closed-loop system is:

$$T(s) = \frac{G(s)G_c(s)}{1 + G(s)G_c(s)} \tag{4}$$

Feedback (Closed-Loop) Controllers Transfer Function

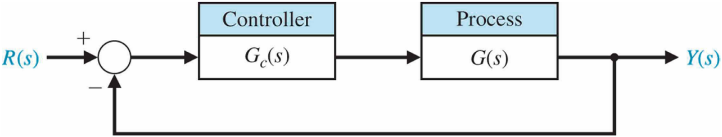


The transfer function of the closed-loop system is:

$$T(s) = \frac{G(s)G_c(s)}{1 + G(s)G_c(s)} \tag{4}$$

- A system is unstable where the closed loop transfer function diverges for s (e.g. where $G(s)G_c(s) = -1$).

Feedback (Closed-Loop) Controllers Transfer Function



The transfer function of the closed-loop system is:

$$T(s) = \frac{G(s)G_c(s)}{1 + G(s)G_c(s)} \tag{4}$$

- A system is unstable where the closed loop transfer function diverges for s (e.g. where $G(s)G_c(s) = -1$).
- Stability is guaranteed when $G(s)G_c(s) < -1$.

PID Control for the Robot Arm Surgeon

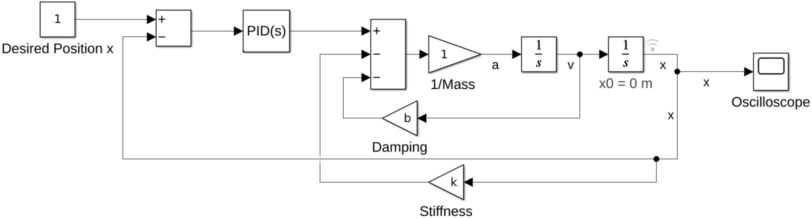
$$m\ddot{x} + b\dot{x} + kx = f \tag{5}$$

The desired position is 1, starting at position $x = 0$. For all the simulations, the following parameters were used:
$m = 1, b = 6, k = 9.86960$.

PID Control for the Robot Arm Surgeon

$$m\ddot{x} + b\dot{x} + kx = f \tag{5}$$

The desired position is 1, starting at position $x = 0$. For all the simulations, the following parameters were used:
$m = 1, b = 6, k = 9.86960$.

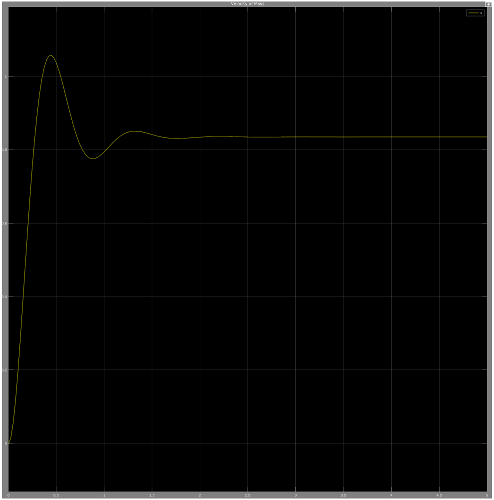


P-Controller

$$K_p = 50$$

P-Controller

$K_p = 50$



P-Controller (cont'd)

- large steady-steady error

P-Controller (cont'd)

- large steady-steady error
- large overshoot

P-Controller (cont'd)

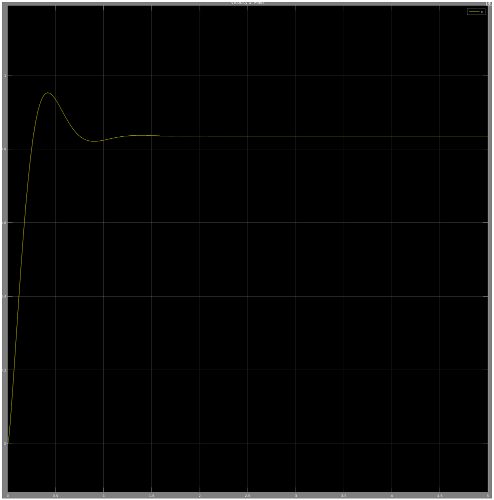
- large steady-steady error
- large overshoot
- large settling time

PD-Controller

$K_p = 50, K_d = 2.5$

PD-Controller

$K_p = 50, K_d = 2.5$



PD-Controller (cont'd)

PD-Controller (cont'd)

The *D*-component has reduced:

- the overshoot

PD-Controller (cont'd)

The *D*-component has reduced:

- the overshoot
- the settling time

PD-Controller (cont'd)

The *D*-component has reduced:

- the overshoot
- the settling time

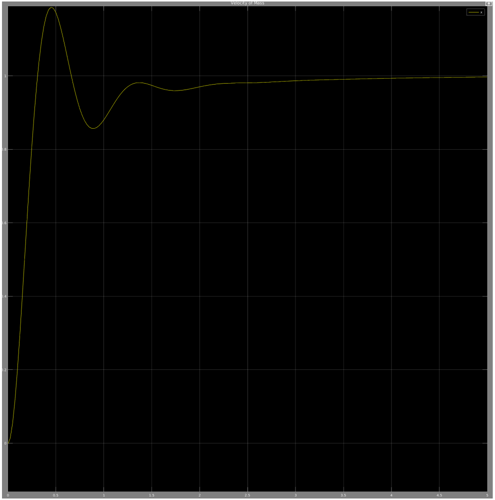
Still large steady-state error.

PI-Controller

$K_p = 50, K_i = 40$

PI-Controller

$K_p = 50, K_i = 40$



PI-Controller

PI-Controller

- 0 steady-steady error

PI-Controller

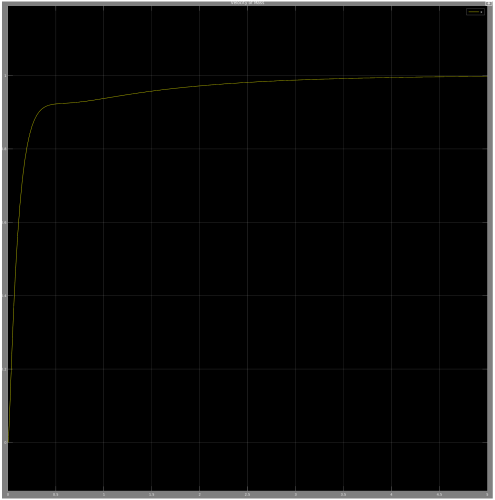
- 0 steady-steady error
- still large overshoot

PID-Controller

$K_p = 50, K_i = 40, K_d = 8$

PID-Controller

$K_p = 50, K_i = 40, K_d = 8$



PID-Controller (cont'd)

- no steady steady error
- no overshoot
- faster rise time

Implementing a Dynamic System and a Controller Programmatically (not Simulink)

Car cruise control can be found in most of the modern cars. The cruise control system keeps the car at a constant speed despite any external disturbances, such as the road surface and incline.

- The car's mass m is controlled by a force u. To simplify the situation, we assume that the force u that our controller applies is not affected by other parameters such as tires, etc.

The equation of the system is described by the following:

$$m\dot{v} + bv = u \tag{6}$$

where v is the speed of the car, u is the control action and b is the damping coefficient due to friction.

The Car Cruise Control System

$$m\dot{v} + bv = u \tag{7}$$

- $m = 1000, b = 50$.
- The desired (reference) speed is $v_{ref} = 10$.
- $t = 10.0$ is the total simulation time
- The parameters of the PID controller are:
  $K_p = 800, K_i = 0, K_d = 40$ (i.e. this is a PD controller).

Implement in Java a PID controller to bring the system to the desired speed.

The Car Cruise Control System (cont'd)

```
import java.io.*;

class CruiseControl {
    static double v = 0;  // current speed initialised to 0
    static double previous_v = 0;  // the speed at the previous time step
    static double dt = 0.001;  // time step for the simulation


    /* Implementnts the dynamic system (plant) - system input is action u
       and the method returns the output of the plant */
    static double plant(double action_u) {
        // m \dot{v} + b v  = u
        // m=1000, b=50, u = 500
        double m = 1000.0;
        int b = 50;

        double v_dot = (action_u - b*v)/m;

        double new_speed = v + v_dot*dt;
        return new_speed;
    }
```

The Car Cruise Control System (cont'd)

```
public static void main(String[] args) {
    PrintWriter pw = null;

    try {
        pw = new PrintWriter("myfile.txt");
    }
    catch(Exception ex) {
        ex.printStackTrace();
    }

    double start_time = 0;
    double end_time = 10.0;

    double current_time = start_time;

    double v_ref = 10;  // the desired speed

    int K_p = 800;  // proportional gain
    int K_i = 0;    // integral gain
    int K_d = 40;   // derivative gain

    double previous_error = 0;
    double integral = 0;
```

The Car Cruise Control System (cont'd)

```
/* simulate the system operation from the beginning till
   the end of the simulation */
while (current_time <= end_time) {
    double error = v_ref - v;

    // I(ntegral) component of the PID controller
    integral = integral + error*dt;

    // D(erivative) component of the PID controller
    double deriv = (error - previous_error)/dt;

    // the output (action) of the PID controller
    double action = K_p*error + K_i*integral + K_d*deriv;

    // remember the last error when the previous action
    // was applied to the plant
    previous_error = error;

    // apply the new action to the plant to calculate
    // the new (current) speed
    v = plant(action);
```

The Car Cruise Control System (cont'd)

```
            System.out.println("Time: " + current_time + " Action: " +
                                    action + ", Speed=" + v);
            pw.println(v + " " + current_time);

            // advance the time
            current_time += dt;
        }

        pw.close();
    }
}
```