UNIVERSITY OF VICTORIA, COMPUTER SCIENCE DEPARTMENT

CSC-586A Self-Adaptive Systems – Assignment 2 Part II

Ernest Aaron, Simar Arora, Harshit Jain, Stephan Heinemann

June 19, 2015

1 Part IIa – PID Controller Design

1.1 Resource Control Problem

Our resource problem can be identified as a queueing model which revolves around a server. The resource to be managed is the number of tasks to be admitted to a server which features a certain service time per task. The server may process tasks in parallel depending on its configuration. If the inter-arrival time of tasks becomes too short, the load on the server increases and vice versa. A configurable First-In First-Out (FIFO) queue (e.g. queue size) stores tasks that have to wait for processing while the server is busy. The goal is to keep the number of waiting tasks at a particular theshold.

The example is shown in Figure 1.1 below and demonstrates how the queue size for a server is kept in limits by a Proportional Integral Derivative (PID) controller that regulates the release of tasks to be completed by the server. Simulink provides all building blocks of the system that can be configured as desired (e.g., queue size, concurrency level, service time).

The first plot shows how the governed inter-arrival time of tasks to be processed stabilizes. The second plot shows how the number of tasks in the queue, which shall be kept a level of 10, stabilizes as well.

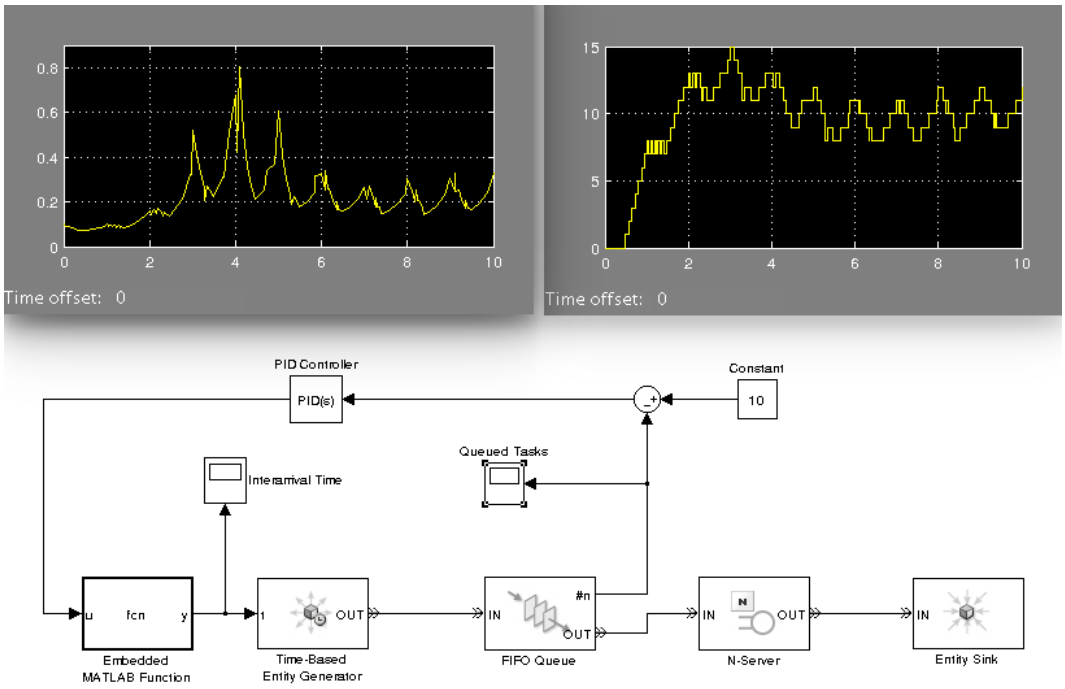


Figure 1.1: Simulink Server Load

The following tutorial illustrates how to model such resource control systems using Matrix Laboratory (MATLAB) and its Simulink component as an extension. We decided to use this extension for our more complex model.

2 Part IIb – PID Tutorial

2.1 System Identification

The first step to tune (design) a PID controller in MATLAB, would be to identify the system to be controlled. In system identification, you need to find the properties including governing equations, inputs and outputs of your system. From these characteristics you can create mathematical models that represent your system (plant). Take for example cruise control: The basic purpose is to maintain the velocity of a car at a constant.

The car may experience various disturbances such changes in slope, air drag and possibly rolling resistance. The introduction of these disturbances can cause the car to speed up or slow down when going uphill. We see that we need to employ a mechanism that measures the desired versus the actual velocity and generates a feedback which aids in keeping the velocity error minimal regardless of disturbances. Figure 2.1 depicts a very simplified model of how the velocity of a car changes [1]. This model only considers acceleration by the driver and deceleration due to friction which increases with increasing velocity.

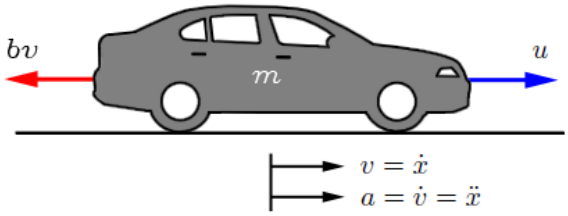


Figure 2.1: Cruise Control Schematic

We can model the change in velocity by the acting forces on the car. The force required for acceleration is $F_a = m * a = m * \ddot{x}$ where m represents the vehicle mass and a its acceleration, respectively. The force that causes the car to decelerate is $F_d = v * b = \dot{x} * b$ where v represents the current velocity and b a friction constant that causes a deceleration. The input $u(t)$ to the system is, hence, the sum of the acting forces where the output $y(t)$ is the resulting velocity.

$$u(t) = m\ddot{x} + b\dot{x},\ y(t) = \dot{x}$$

2.2 Laplace Transformation and Transfer Function

Once the system has been identified and the plant has been described by the governing equations, its transfer function needs to be established. The transfer function of a system is defined as the relation between its output and input state

$$P(s) = Y(s)/U(s)$$

[1] http://ctms.engin.umich.edu/CTMS/index.php?example=CruiseControl&section=SystemModeling

where $P(s)$ is the plant transfer function, $Y(s)$ is the output state and $U(s)$ the input state of the plant. Most (cyber-) physical system equations depend on time and system behavior is usually described with respect to time. Hence, the equations usually describe states and their derivatives with respect to time such as position, speed and acceleration in the cruise control example.
In order to establish the transfer function of a plant, its governing equations need to be transformed into the frequency (s) domain. The Laplace transform of a function $f(t)$ is defined as the improper integral.

$$\mathscr{L}\{f(t)\} = \int_0^\infty e^{-st}\,dt$$

One of the most important rules for differential equations is the Laplace transform of derivatives that are often found in plant equations

$$\mathscr{L}\{f'(t)\} = s\mathscr{L}\{f(t)\} - f(0)$$

where $f(0)$ represents the initial condition with respect to f at time 0.
Applying the Laplace transform and the above rule to the cruise control example yields

$$P(s) = Y(s)/U(s) = \mathscr{L}\{\dot{x}\}/\mathscr{L}\{\, m\ddot{x} + b\dot{x}\} = \mathscr{L}\{\dot{x}\}/ms\mathscr{L}\{\dot{x}\} + b\mathscr{L}\{\dot{x}\} = 1/(ms+b)$$

The resulting transfer function of the plant (e.g., $1/(ms+b)$ in the cruise control example) can then multiplied to other transfer functions in the control loop in order the reduce the entire system to a single transfer function as desribed in Section 2.3. Instead of solving the Laplace transformation by hand, we could also reference the applicable Laplace tables.

2.3 Matrix Laboratory

After identifying your system and deriving your plant transfer function you can now move on to assembling and simulating your system in MATLAB.

2.4 Cruise Control Parameters

Firstly, you need to define the constants for your modeled plant – in this case the cruise control system. The parameters are as follows:

Vehicle Mass $m = 1000kg$
Damping Coefficient $b = 50Ns/m$
Reference Speed $r = 10m/s$

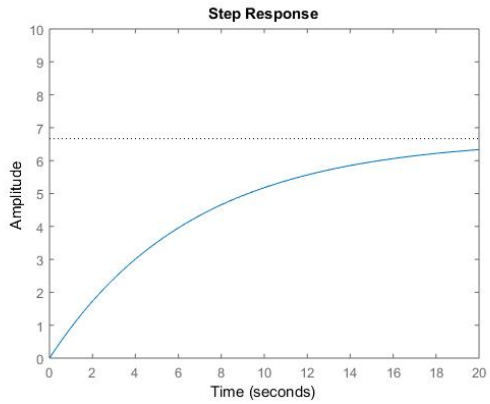
2.5 MATLAB IMPLEMENTATION

Using the following sequence of commands, you establish a feedback control loop with the car modeled above being the controlled plant.

```
s=tf('s'); % define the basic transfer function s
m=1000; % define the vehicle mass to be 1000kg
b=50; % define the damping coefficient to be 50Ns/m
r=10; % define the reference speed to be 10m/s
G=1/(m*s + b); % define the plant (car) transfer function
Kp=1; % define the proportional gain
Ki=0; % define the integral gain
Kd=0; % define the derivative gain
C=pid(Kp,Ki,Kd); % construct the PID controller using the gains
T=feedback(C*G,1); % construct the feedback loop with controller and plant
t = 0:0.1:20; % define the time interval for simulation
step(r*T,t) % plot the step response of the system in the time interval
```

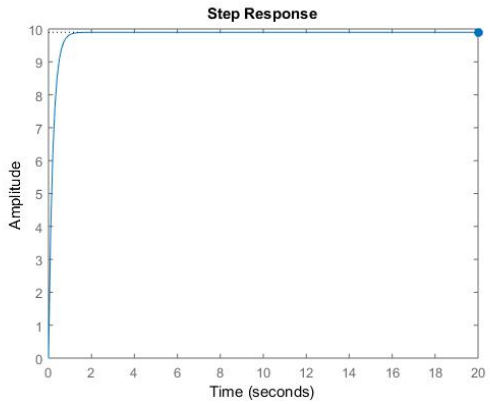
The above sequence employs the built-in PID controller object which already abstracts the PID transfer function $C = K_p + K_i/s + K_d s$ which you could also define manually. The sequence furthermore uses the feedback object multiplying all transfer functions in the control loop, that is, $C * G$. Recall that our transfer function for the car is $G = 1/(ms + b)$ as derived in section 2.2.

In general you may want to start your simulation by adjusting the proportional gain only and then proceed with the remaining gains. Now you can adjust the values of Kp, Ki and Kd to achieve an acceptable system response. If $Kp = 100$ and $r = 10$ (reference speed), the step response is as shown in Figure 2.2a.



Not acceptable for cruise control. The rise time (14.6 seconds) is too high and not acceptable. The steady state error ($10 - 6.67 = 3.33$) is too high and not acceptable.

(a) Cruise Control Step Response - $K_p = 100$



Not acceptable for cruise control. The rise time (0.445 seconds) is too short and not acceptable. The steady state error is now approximately zero and acceptable.

(b) Cruise Control Step Response - $K_p = 4900$

Figure 2.2: Cruise Control Step Responses I

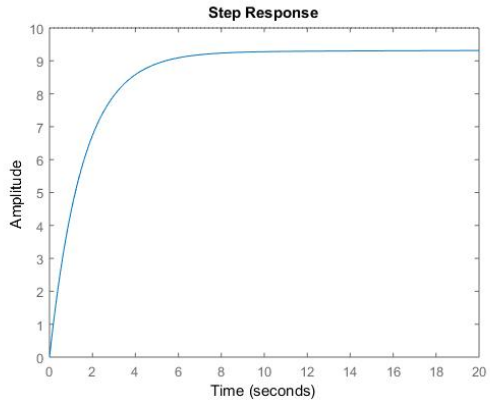
The important properties of the step response are *rise time, overshoot* and *settling time*. You can get these

properties from the generated plot by right-clicking on the graph and selecting the *Characteristics* context menu item.

So we have to increase the value of Kp (proportional gain) because it reduces the steady state error as well as the rise time but up to a certain limit. Higher values of Kp lead to overshoot which is not acceptable for the cruise control system because it leads to oscillations. Short rise times are also limited by the acceleration performance of the car.

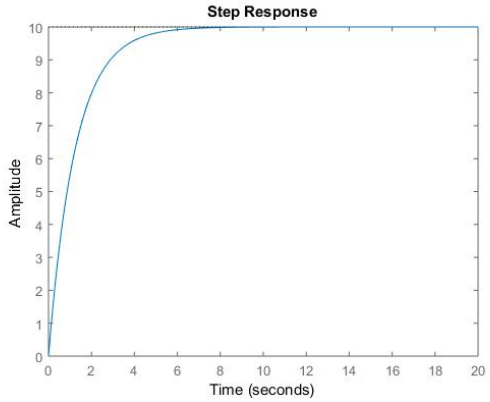
If $Kp = 4900$ and $r = 10$ (reference speed), the step response is as shown in Figure 2.2b. Now the system features a very short rise time of about 0.445 seconds which is not practically acceptable for the system because the car is unlikely to accelerate from 0 to $10 m/s$ in just 0.445 seconds.

So we have to add the integral term in the controller and reduce the value of the proportional gain so its gives the acceptable rise time and the integral term will eliminate the steady state error. When $Kp = 600$, $Ki = 2$ and $r = 10$ (reference speed), the step response is as shown in Figure 2.3a.



Less acceptable for cruise control. The settling time is very high and not acceptable. The steady state error is now zero and acceptable.

(a) Cruise Control Step Response - $K_p = 600, K_i = 2$



Acceptable for cruise control. The settling time (4.89 seconds) is acceptable. The rise time (2.75 seconds) is acceptable. The steady state error is now zero and acceptable.

(b) Cruise Control Step Response - $K_p = 800, K_i = 10$

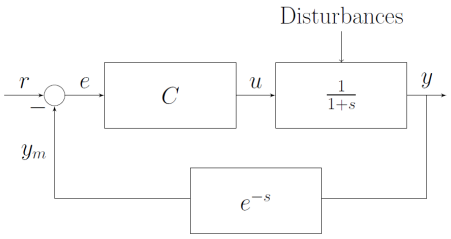
Figure 2.3: Cruise Control Step Responses II

Now the system is a little more acceptable but the settling time is still very high. So we have to adjust both gains (proportional and integral) in such a way that they result in the desired response. We have to start with a lower value of Ki because a higher value will destabilize the system. If $Kp = 800$, $Ki = 10$ and $r = 10$ (reference speed), the step response is as shown in Figure 2.3b.
Now the system is acceptable possessing a suitable settling time, rise time and zero steady state error. We do not need to add the derivative term to the controller because we are satisfied with the PI controller.

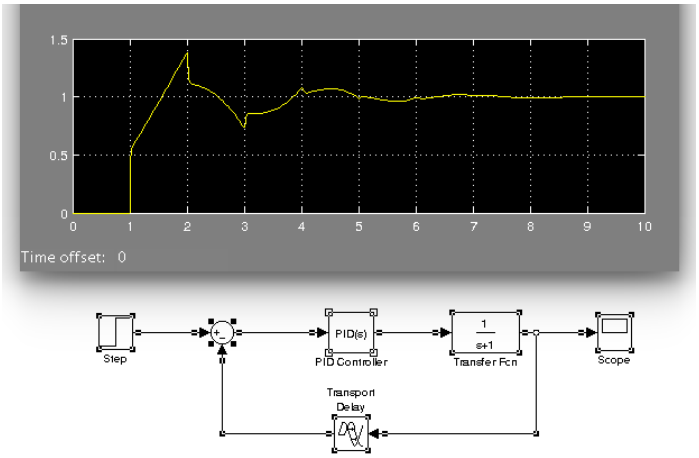
2.6 SIMULINK

More often than not we may need to model systems that are highly complex or in hindsight the equation is not known. Simulink provides a graphical programming environment that allows us to use components that already have the transfer function built into them. Simulink can be accessed through MATLAB

by typing the command simulink. If you are given the following Linear Time-Invariant (LTI) system as in Figure 2.4a, without knowing the MATLAB command line statements, as demonstrated in Section 2.3, you could open Simulink and replicate the following system.



(a) LTI System



(b) Simulink LTI System

Figure 2.4: LTI System Model

To start, select the components that represent your system from the Simulink Browser Library (see Figure 2.5). These could include sources (e.g., steps), transfer blocks (e.g., functions) and sinks (e.g., scopes) among many other types of building blocks.

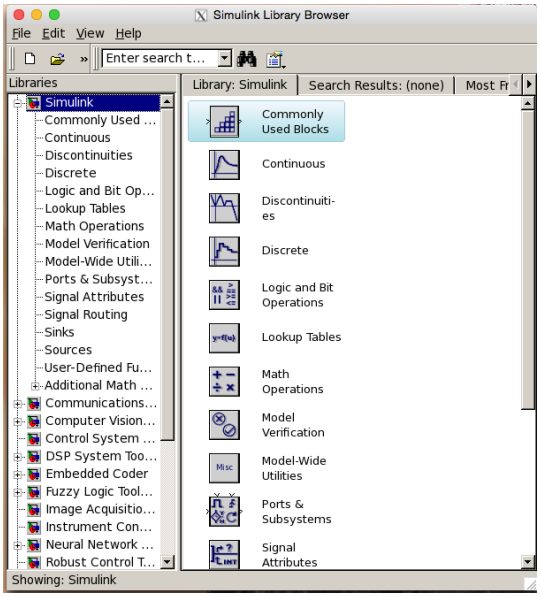


Figure 2.5: Simulink Library Browser

Provided that you have assembled your system correctly, you can simulate it by clicking the Simulation/-Start menu item. Figure 2.4b depicts the above LTI system and simulation result in Simulink using the Ziegler-Nichols method for the PID controller gains.

3 Group Members

csc586a-ernest-aaron-v00815728
csc586a-simar-arora-v00824821
csc586a-harshit-jain-v00827753
csc586a-stephan-heinemann-v00814821

4 ACRONYMS

FIFO First-In First-Out First-In First-Out is a method for organizing and manipulating a data buffer, where the oldest (first) entry, or 'head' of the queue, is processed first. [2]

LTI Linear Time-Invariant Linear time-invariant theory comes from applied mathematics and has direct applications in NMR spectroscopy, seismology, circuits, signal processing, control theory, and other technical areas. It investigates the response of a linear and time-invariant system to an arbitrary input signal. [3]

MATLAB Matrix Laboratory Matrix Laboratory is a multi-paradigm numerical computing environment and fourth-generation programming language. [4]

PID Proportional Integral Derivative A proportional-integral-derivative controller is a control loop feedback mechanism (controller) widely used in industrial control systems. [5]

[2] https://en.wikipedia.org/wiki/FIFO_computing_and_electronics
[3] https://en.wikipedia.org/wiki/LTI_system_theory
[4] https://en.wikipedia.org/wiki/MATLAB
[5] https://en.wikipedia.org/wiki/PID_controller

References