Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
Institut für Dynamische Systeme und Regelungstechnik

151-0593-00 **Embedded Control Systems** (Fall 2017) **Lab 6**

Topic: **Virtual Worlds with Dynamics**

Pre-Lab due: 19. 09. 17 at 1 p.m. Post-Lab due: 19. 09. 17 at 5 p.m.

Name: .... Forename: .... Initials: ....

marianne.schmid@idsc.mavt.ethz.ch, 19. September 2017

1 Overview

In this lab, we will design and implement a number of simple virtual worlds. These worlds are various connections of static positions (walls), dampers (dashpots), and springs. Our interface to the virtual worlds is the haptic wheel (the puck), which we used previously. The haptic wheel is a rotational device that can apply torque to the user's hand. We draw the systems below as translational systems only for simplicity.

The fundamental difference between these virtual worlds and those of Lab 4 is that we now require a notion of time, which will be supplied by the interrupt timer (DEC), which we learned to use in Lab 5. Unless stated otherwise, we will execute all code in this lab in a timed loop with $T = 1$ ms. We shall also use the functions for quadrature decoding and Pulse Width Modulation (PWM) that we developed in earlier labs.

1.1 Virtual Wall with Damping

This system is similar to the virtual wall from Lab 4. We will add damping to the virtual wall to reduce the chatter felt with high spring constants. See "Notes on Wall Chatter" on the course website, which was discussed in the lecture. Recall the "ideal" value of damping, $b_{wall} = kT/2$ that compensates for the half sample delay due to the ZOH. We will test the system with this ideal value and various other values.

1.2 Virtual Spring with Damping

This system, depicted in Figure 1, is similar to the virtual spring from Lab 4, except that the wheel is now connected to the reference position with a damper in addition to the spring. The purpose of the damper is to provide a retarding torque proportional to the angular velocity of the wheel: $\tau_d = -b\frac{d\theta_z}{dt}$. This torque is in addition to the restoring torque of the spring: $\tau_s = -k(\theta_z - \theta_z^*)$. Recall there is an ideal value of damping, b_{spring} that makes the resulting system critically damped: b_{spring} is the smallest value of b such that the wheel, after release, will return monotonically to its reference position. A formula for the value of b_{spring} is given in section 4 in the handout "Implementation Issues for the Virtual Spring", discussed in class and available on the website.

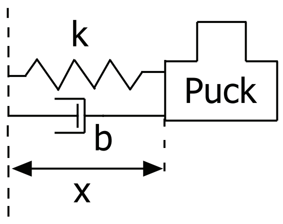

Figure 1: Virtual Spring with Damping

In both above-mentioned applications, the motor must supply a torque equal to the sum of the virtual spring and virtual damper torques. To compute the latter, we need a measure of the velocity of the wheel.

An effective way to determine velocity is through numerical differentiation. By setting the DEC so that an interrupt occurs every T seconds, velocity may be approximated by $\Delta\theta_z/\Delta t = (\theta_z(current) - \theta_z(old))/T$.

1.3 Virtual Spring Mass System

Next consider the virtual spring mass system depicted in Figure 2, which is described in the handout "The Virtual Spring Mass System" (see course website). In this system, it is assumed that the position of the puck (in our case, the haptic wheel) is controlled by contact with the hand of the user, so that the resulting system has dynamical order two, with an input given by the position of the puck.

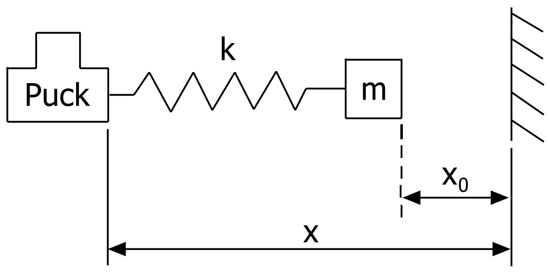

Figure 2: Virtual Spring Mass System

As discussed in class, implementing the virtual spring mass system as shown in Figure 2 is difficult, because the system is an ideal harmonic oscillator, and the destabilizing effects of Forward Euler integration will make the virtual world implemented on the MPC5553 unstable. This may be seen by observing the resulting PWM signal on the oscilloscope – **do not apply** this signal to the motor!

1.4 Virtual Spring Mass System with Damping

To compensate for the destabilizing effect of Forward Euler integration, we can add virtual damping b to the virtual spring mass system, as shown in Figure 3. By setting $b = b_{osc} = kT$, the value derived in "The Virtual Spring Mass System", the destabilizing effects of the Forward Euler integration are canceled exactly, and the virtual world implemented on the MPC5553 is indeed a harmonic oscillator.

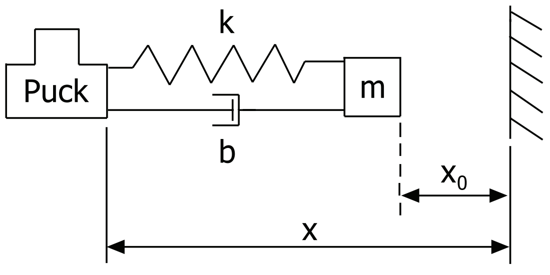

Figure 3: Virtual Spring Mass System with Damping to achieve numerical stability

Note: There is an important conceptual difference between the values of damping computed in Subsections 1.1 and 1.2 and that computed in Section 1.4. In the latter case, the damping $b = kT$ is the precise value required to offset the effects of Euler integration, and will result in a simulation that is precisely a discrete-time harmonic oscillator. In the former cases, the values of damping added are only approximate, as they are intended to compensate for complex effects such as the energy leak for the virtual wall and for the underdamping inherent in the wheel/motor combination to which the virtual spring and damper are attached. As a result, one would not expect these values to have precisely the intended effect, as they were derived from simplified models.

1.5 Virtual Knob

A virtual knob is an implementation of a mechanical selector knob with detents that can be felt at regular intervals, as you might find on a car stereo or home theater receiver. Read the TouchSense [1] document provided on the course website to familiarize yourself with the uses of programmable haptic effects.

2 Design Specifications

2.1 Hardware

As we learned in Lab 4, the haptic interface available in-lab contains a power supply, an amplifier, an encoder, a motor, and two ribbon cables connecting the amplifier and the encoder to the interface board. The power supply converts 230 VAC into 24 VDC to be used by the amplifier. The amplifier receives a low-pass filtered PWM signal (eMIOS0) from ANout0 and drives the motor. The quadrature encoder signals are eTPUA3 and eTPUA4. The amplifier receives the demodulated PWM signal as a voltage command and drives the motor with a current command that is proportional to the desired torque.

2.2 Software

You will need to use your FQD, PWM, and ISR software from the previous labs to control the haptic interface. Note that the PWM output needs to be **limited to 24 to 76% duty cycle** to prevent damage to the haptic devices. The system clock frequency is set to 120 MHz.

Your *lab6.c* file should have your main function and call the necessary initialization functions to set up your virtual world. A majority of numerical processing will take place in the ISR called by the DEC, not in the main function.

Interrupt requests are produced at a fixed rate. Your ISRs should determine the position and velocity of the haptic wheel, call the virtual world function with this information, and then output the torque to the motor. Note that the virtual world functions are in fact transfer functions from haptic position and velocity to torque.

For the spring and wall-damper, and the spring-mass-damper systems, you will need to use the DEC to keep track of time. Use the provided makefile to compile your code.

3 Pre-Lab Assignment

Pre-lab questions must be done **individually** and handed in at the start of the in-lab section. You must also, **with your partners**, design an initial version of your software before the in-lab section.

1. Using the formula from the section 4 of the handout "Implementation Issues for the Virtual Spring", calculate the amount of virtual damping required to make the virtual spring critically damped. Find the numerical solution in Nmm/(deg/s) using $J = 4.5 \cdot 10^{-4}$ Nm/(rad/s^2), $b_m = 0.0013$ Nm/(rad/s) and $k = 50$ Nmm/deg. **Be careful with the units.**

2. Implement the **Virtual Spring with Damping**. This function will be called periodically by the DEC ISR to calculate and apply the appropriate torque response to the system at that time. The DEC will need to be used in order to develop a concept of time in the microcontroller. The velocity calculation will be done in the ISR, not in the virtual world function.

   Follow the function prototype for `virtualSpringDamper`, as provided in the *worlds.h* file, and implement the function in the *worlds.c* file. The two input arguments represent the current angle and current velocity of the wheel.

3. Implement the **Virtual Wall with Damping** and calculate the amount of virtual damping to reduce wall chatter due to the half-sample delay of the ZOH. Use the best value of k that you found for the virtual wall in Lab 4, and express your answer in units of Nmm/(deg/sec).

4. Implement the **Virtual Spring Mass System** and the **Virtual Spring Mass System with Damping** by performing the following steps. Use the equations developed in class for state space approximations of the differential equations.

(a) First, calculate the torsional spring constant k and the rotational inertia J_w so that the following properties are satisfied: Suppose that you turn the wheel 45 degrees and hold its position constant. Then the virtual mass should oscillate with a 1 Hz frequency, and the maximum torque generated should be 800 Nmm. Provide the numerical solutions using degrees as unit for the angles.

(b) Verify with a MATLAB simulation that the values you chose give the correct behavior, and include a response graph in your pre-lab. You may use the provided MATLAB files and use *virtual_wheel_noJbk.m* as a starting point.

(c) Second, write down a discrete model of the virtual spring mass system, using Forward Euler integration to approximate the analog integrators. Use a 1 ms integration step size. Calculate the closed-loop characteristics roots (i.e. closed-loop poles) for the system. Express your answer numerically. You may find lecture notes 8 helpful.

Implement the **Virtual Spring Mass System** in *worlds.c*.

(d) Third, compute the precise amount of damping that must be added to the virtual spring mass system in order to implement a stable harmonic oscillator on the MPC5553. Use this information to implement **Virtual Spring Mass System with Damping** in *worlds.c*.

4 In-Lab Assignment

1. Connect the haptic device to the interface board using the encoder input **Enc_2**.

2. Implement the corresponding ISR function and the main function in *lab6.c* and test the **Virtual Wall with Damping**. Vary the amount of damping to see how the level of chatter varies, but be sure to try the "ideal" value calculated in lecture.

3. Implement the corresponding ISR function in *lab6.c* and test the **Virtual Spring with Damping** using a sample period $T = 0.1$ ms and a spring constant $k = 50$ Nmm/deg. Start with zero and increase the damping coefficient b until the motion of the wheel becomes critically damped. That is, the wheel should not oscillate about its final position. Be sure to record this value after you test it several times. **Demonstrate the running system to your teaching assistant.**

4. Recall your **Virtual Spring** which you already implemented in *worlds.c* during Lab 4. In that lab we implemented the spring in a (very fast) untimed loop, and saw that it exhibited oscillations that decayed to zero. The reason for this, as discussed in lecture, is that the inherent damping in the motor is more than sufficient to overcome the destabilizing effects of the sampled-data implementation. Now in Lab 6 we work with software loops that run at a rate $T = 1$ ms. Change the Virtual Spring with Damping ISR to execute the Virtual Spring (without damping). With a sample period of $T = 1$ ms, does the virtual spring still exhibit decaying oscillations? If not, try a $T = 0.1ms$ loop. Note your observations for later discussion.

5. Turn the motor off. After that, implement the corresponding ISR function in *lab6.c* and test the **Virtual Spring Mass System** using a sample period $T = 1$ ms (use this value of T for the remainder of Lab 6) and the spring constant calculated in the Pre-Lab. Observe the unstable ANOut0 signal on the oscilloscope, but **do not apply** this to the motor. Compare the results with your MATLAB simulations and **demonstrate it to your assistant.**

6. Implement the corresponding ISR function in *lab6.c* and test the **Virtual Spring Mass System with Damping**. Before enabling the motor, observe the ANOut0 signal on the oscilloscope to verify stability. Compare the behavior with your MATLAB simulations by providing the damping value to the MATLAB script.

7. Using the header files given in previous labs, develop a terminal interface to the spring mass system with damping. This interface should run as task code in the program while the spring mass calculations will run as an interrupt. With the terminal interface, provide a way to change the spring constant and the mass of the system using the '+' and '-' keys. Adjust the parameters and notice how the system changes. As you change your spring constant and mass values, your damping should also change such that the resulting system is stable. Implement the function `virtualSpringMassDamperEx` in *worlds.c* and the corresponding ISR function in *lab6.c*.

Note, because K and M are shared between `main` and the interrupt service routine `smdIsrEx`, write access to these variables needs to be exclusive. While `main` manipulates the values, `mdIsrEx` must not read the value. Use `ENTER_CRITICAL()` and `LEAVE_CRITICAL()` to disable and re-enable interrupts in `main` while the new value for K or M is being calculated. Make the part between `ENTER_CRITICAL()` and `LEAVE_CRITICAL()` as short as possible because this will delay the execution of the ISR! Example code:

```
#define ENTER_CRITICAL() asm (" wrteei 0"); /* disable all interrupts */
#define LEAVE_CRITICAL() asm (" wrteei 1"); /* enable all interrupts */

void main()
{
    while(1)
    {
        ...
        ENTER_CRITICAL();
        K = K * 1.1;                        /* while main manipulates K
                                             * isr must not read the variable */
        LEAVE_CRITICAL();
        ...
    }
}
```

Demonstrate the running system to your assistant.

8. Implement the **Virtual Knob** and the corresponding ISR which satisfy the following function. Create a world in which a resistive force is felt when turning the haptic wheel, to simulate friction. Because this force is only present when the wheel is moving, should you use a spring force or should you use damping? Use your virtualKnob world in *lab6.c* and test your system to find settings that provide smooth resistance to motion, like you might feel on a volume knob of a stereo receiver.

9. Add virtual detents to the virtual knob. Use the constant friction you developed earlier, but when the wheel reaches certain angles, alter the force such that the user feels a bump or a tick. Configure the detents (the bumps felt at the wheel) to be 10 degrees apart, and **demonstrate the system to your assistant**. Because you have full control over the severity and spacing of the detents, configurable tactile feedback is useful when interfaced with many software applications, as described in [1].

5 Post-Lab Assignment

1. From In-Lab 2, you observed that the addition of damping $b = kT/2$ reduced, but did not completely eliminate, the wall chatter effect in the virtual wall. However, $b = kT/2$ only compensates approximately for the time sampling artifact of the digital implementation. In addition, encoder resolution implies that the reaction torque in response to a change in wheel position can only change in discrete amounts. Neglect the effect of time sampling, and suppose you turn the wheel into the virtual wall. How will the reaction torque vary as you do so? Use the value of encoder resolution you calculated in Lab 2 and the spring constant k proposed for the virtual wall in Lab 4. To answer this question, assume that reaction torque is due only to a virtual spring.

2. From In-Lab 4, how did the virtual spring behave with $T = 1$ ms? with $T = 0.1$ ms? From In-Lab 3, what value of b yielded critical damping for $T = 0.1$ ms? How does this value compare with the theoretical value you computed in Pre-Lab?

3. To test your understanding of the theory for the virtual spring, compute the expected closed loop pole locations for the values $J = 4.5 \times 10^{-4}$ Nm/(rad/sec^2), $b_m = 0.0013$ Nm/(rad/sec), $k = 50$ Nmm/deg, and the sample times $T = 1$ ms and $T = 0.1$ ms. Do these poles lie inside or outside the unit circle? Does this analysis agree with your observations from In-Lab 4?

If you have comments that you feel would help improve the clarity or direction of this assignment, please include them following your post-lab solutions.

References

[1] *TouchSense Programmable Rotary Modules*, Immersion Corporation, www.immersion.com, Lit#BR.TSRotary.1004.1000.v3 (2004)