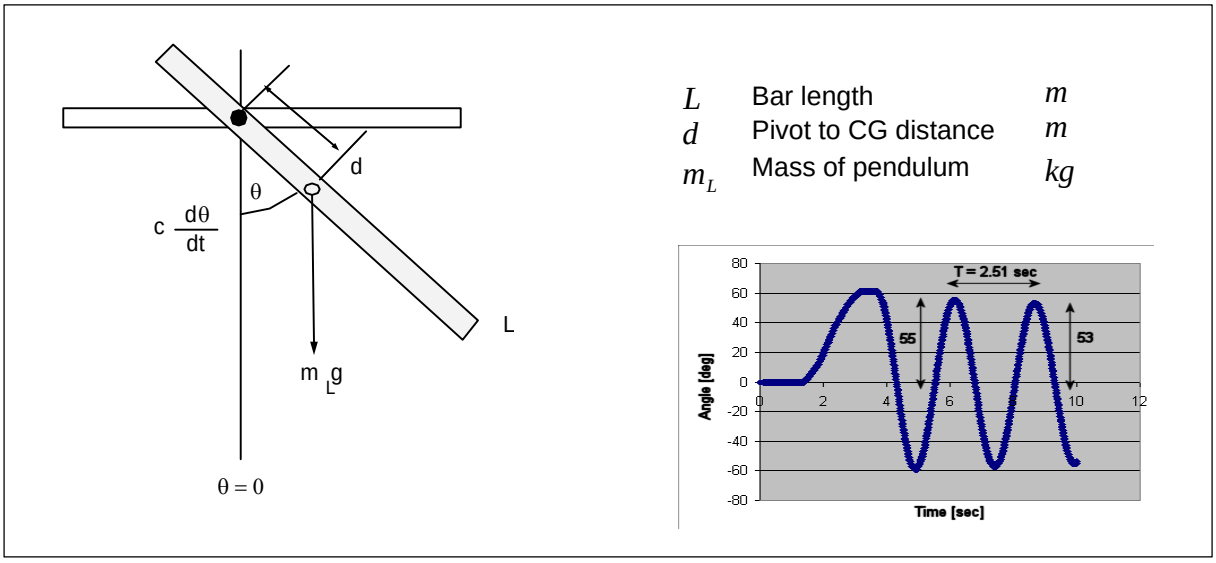
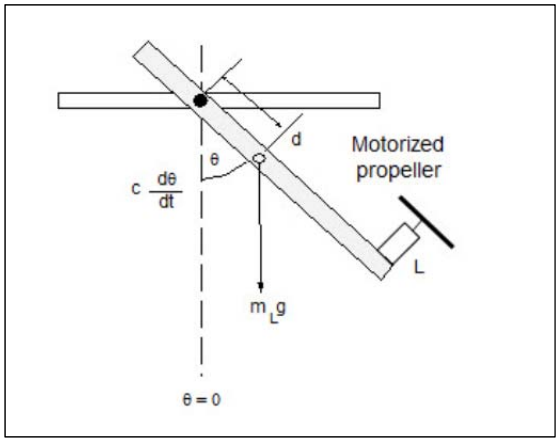
Hands-on Lab

Damped Compound Pendulum – System ID (Experimental and Simulation)

Preamble:



Sketched above is a damped compound pendulum; instead of a string with a mass at the end, the pendulum is made from a solid (*compound*) material. Intuitively, the time response of the pendulum's angle would be a decaying oscillation. Due to *damping*, the angle would eventually decay to $\theta=0$ degrees. Because of the oscillation, this is known dynamically, as a second-order system.



Suppose one attached a motorized propeller at the end of the damped compound pendulum, as shown in the above drawing. When the motor is actuated, the propeller spins and thus generates lift. The damped compound pendulum will move and eventually reach a steady-state angleθ. This angle defines an equilibrium state; the lift balances the gravitational force acting on the damped compound pendulum.

How quickly the damped compound pendulum reaches steady-state depends on the propeller's velocity. To reach steady-state quickly (i.e. fast rise time), one can command the propeller to spin very fast. However, this could lead to a lot of oscillations (i.e. a stability issue). To reduce oscillations, one can command the propeller to spin slowly (i.e. good stability). But then the damped compound pendulum would take a long time to reach steady-state (i.e. slow rise time).

General Objective: Design a controller that regulates the motorized propeller so that the pendulum reaches a desired angle within a desired rise time and desired oscillation (i.e. overshoot). PID control will be implemented to achieve such goals.

Specific Objective: Towards controller design, one must first understand the system's dynamics. This lab will employ an angle sensor to capture the damped compound pendulum's dynamics. This process is called system identification – which reveals the system's characteristic equation and hence the system's inherent stability (i.e. poles) and performance (i.e. zeros).

Do First! See Build Instructions to construct a LEGO-based motorized damped compound pendulum. The motorized propeller is driven by a DC motor driver (MTRMX-Nx). This motor driver is sold by a 3rd-party vendor (Mindsensors) and is NXT-compatible. The pendulum's state is measured by an angle sensor made by HiTechnic.

Concept 1 System Identification (Experimental)

1-1: Measuring Pendulum angle

Step 1: After assembling your motorized damped compound pendulum, interface the NXT brick to the angle sensor. See photo below for reference.

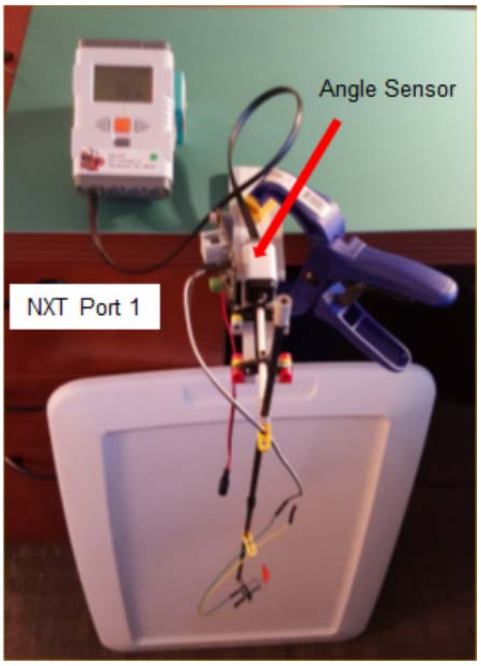


Fig. 1-1A: The HiTechnic angle sensor is connected to the NXT brick on Port 1

Step 2: Enter, compile and execute `helloHtAngleSensor1_0.nxc`. Observe NXT screen.

The HiTechnic angle sensor uses I2C to interface with an NXT brick. HiTechnic's angle sensor reports measurements as an integer ranging from 0 to 359. Thus, this sensor has only 1-degree accuracy.

```
// DESC: HiTechnic Angle Sensor on Brick Port 1.  Display sensor data on LCD
// VERS: 1.0: Damped Compound Pendulum.  CCW from rest yields +'ve angle
//            and CW from rest yields -'ve angle

#define ANGLE S1  // HiTechnic Angle Sensor in Brick Port 1

task main() {
  int angle;           // By default, with one's eyes on angle sensor axle,
  int pendulumAngle;   // CW rotation increases from 0 to 359 degrees and
                       // CCW rotation decreases from 359 downwards...
  // For damped compound pendulum at rest position i.e. hanging straight down
  // one wants CCW to increase from 0 towards 90 degrees...
  // Hence define variable pendulumAngle;

  long acc_angle;
  int rpm;
  bool orangeButtonPushed;

  SetSensorLowspeed(ANGLE);
  Wait(100);

  TextOut (0, LCD_LINE2, "Calibrate:" );
  TextOut (0, LCD_LINE4, "Set zero angle" );
  TextOut (0, LCD_LINE6, "Orange Btn: starts" );

  do {
     orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
     Wait(100);  // delay needed for button debounce
     // Nothing until Orange button pushed
  } while(!orangeButtonPushed);

ResetSensorHTAngle(ANGLE, HTANGLE_MODE_CALIBRATE);
  // Angle should be zero now...

  // ButtonPressed(BTNCENTER, TRUE); // reset to false

  ClearScreen();
  TextOut (0, LCD_LINE2, "Running..." );
  TextOut (0, LCD_LINE4, "Orange Btn: Quit" );

  pendulumAngle = 0;  // i.e. damped compound pendulum hanging down at rest
```

```
do {
     Wait(100);
     ReadSensorHTAngle(ANGLE, angle, acc_angle, rpm);

     if(angle >= 180 && angle <= 360)
        // pendulum rotating CCW, so pendulumAngle increases positively from 0
        pendulumAngle = 360 - angle;

     if(angle >=0 && angle < 180)
        // pendlum rotation CW, so pendulumAngle increases negatively from 0
        pendulumAngle = -angle;

     TextOut(0, LCD_LINE8, FormatNum("Angle = %3d deg" , pendulumAngle));
     orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
  } while(!orangeButtonPushed);

  // Orange button pressed, so command 0 speed to motor
      ClearScreen();
      TextOut(0, LCD_LINE2, "Quitting", false);
  PlaySound(SOUND_LOW_BEEP);  // Beep to signal quitting
      Wait(SEC_2);

} // end main
```

Code Explanation: The HiTechnic angle sensor is connected to the NXT brick through Port 1. As such, a variable `ANGLE` is defined as `S1`. The angle sensor is then initialized with a `SetSensorLowSpeed (ANGLE)` statement. A few variables are declared to report raw angle, calculate pendulum angle and detect the state of the NXT orange button.

The `ResetSensorHTAngle(ANKLE, HTANGLE_MODE_CALIBRATE)` statement will read the sensor and set the value to 0. In other words, this allows one to calibrate the angle.

Suppose the following reference frame is desired: Let $\theta = 0$ to be the angle when the pendulum is motionless (i.e. hanging down); counter-clockwise angles yield $0 \leq \theta \leq 180$; and clockwise motions are $-180 \leq \theta \leq 0$. To variable `pendulumAngle` is used to reflect such a reference frame.

A `do-while` loop reads the angle sensor using a `ReadSensorHTAngle(ANGLE, angle, acc_angle, rpm)` statement, calculates `pendulumAngle`, and displays the value on the LCD. The loop exits when the NXT's orange button is pushed.

Exercise

1.1 Modify helloHtAngleSensor1_0.nxc to reflect the following reference frame: 0 degrees is when the pendulum is horizontal (i.e. 3 o'clock position), 90 degrees is 6 o'clock and 180 degrees is at 9 o'clock.

1-2: Capturing Pendulum Data and identifying characteristic equation

Step 1: Download and compile `helloHtAngleSensorSaveToFile1_0.nxc`

The code `helloHtAngleSensorSaveToFile1_0.nxc` measures elapsed time and writes captured angle data to a file named **`myAngleData.csv`**.

Step 2: Execute `helloHtAngleSensorSaveToFile1_0.nxc`. Calibrate the pendulum such that 0 degrees is when the pendulum is hanging down motionless (i.e. 6 o'clock position). Then, rotate the pendulum counter-clockwise to the 3 o'clock position (i.e. about 90-degrees in our original reference frame).

Step 3: From Bricx NXT Explorer, copy the capture angle data (`myAngleData.csv`) to your computer. Use Excel to line plot a graph of angle [deg] (y-axis) over time [sec] (x-axis). **Fig 1-2A** shows an example

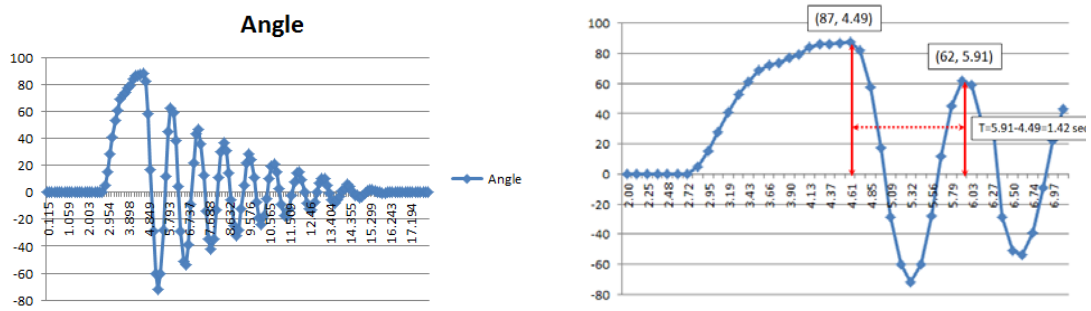


Fig 1-2A: Time plot of pendulum angle in degrees (left). A close up of the time plot (right)

Fig 1-2A (left) is a classic example of a 2nd order dynamic system. At around 3 seconds, the pendulum is moved from 0-degrees to about 90-degress and then released. At about 16 seconds, the pendulum is motionless, settling at 0-degrees.

Theory: One can recall that a 2nd order dynamic system is characterized by the ratio of subsequent peaks and the period. All second order systems have the following form:

$$\ddot{\theta} + 2\zeta\omega_n\dot{\theta} + \omega_n^2 = 0 \tag{1}$$

Where there are 2 relationships:

$$ln\frac{a}{b} = \frac{\zeta 2\pi}{\sqrt{1-\zeta^2}} = \frac{1}{N} ln\frac{X_1}{X_{N+1}} \tag{2A}$$

$$\frac{2\pi}{T} = \omega_n\sqrt{1-\zeta^2} \tag{2B}$$

Fig 1-2A (right) is a close up of the time response and identifies the following: $a = 87, b = 62, \text{and } T = 1.42 \text{ sec}$. Substituting these values into (2A), one has:

$$ln\frac{87}{62} = 0.34 = \frac{\zeta 2\pi}{\sqrt{1-\zeta^2}}$$

$$\left(0.34\sqrt{1-\zeta^2}\right)^2 = (\zeta 2\pi)^2$$

Solving for ζ one has $0.11(1-\zeta^2) = \zeta^2 4\pi^2$ or $0.11 - 0.11\zeta^2 = 39.5\zeta^2$ or ultimately:

$$39.61\zeta^2 = 0.11 \text{ or } \boldsymbol{\zeta = 0.053} \tag{3A}$$

Substituting the damping ratio ζinto (2B), yields the following

$$\frac{2\pi}{1.42} = \omega_n\sqrt{1-0.053^2}$$
$$4.42 = \omega_n\sqrt{0.999} \text{ or } \boldsymbol{\omega_n = 4.42} \tag{3B}$$

Substituting (3A) and (3B) into the general 2nd order dynamic equation (1) yields

$$\ddot{\theta} + 2(0.053)(4.42)\dot{\theta} + (4.42)^2$$

$$\ddot{\theta} + 0.47\dot{\theta} + 19.5 = 0 \tag{4}$$

Recall, that the general solution for (4) is given by

$$\theta_c(t) = e^{-\zeta\omega_n t}\left\{A_1\cos\left(\omega_n t\sqrt{1-\zeta^2}\right) + A_2\sin\left(\omega_n t\sqrt{1-\zeta^2}\right)\right\} \tag{5A}$$

The complex roots are given by

$$s_{1,2} = -\zeta\omega_n \pm j\cdot\omega_n\sqrt{1-\zeta^2} \tag{5B}$$

Substituting the values from (3A) and (3B) into (5B), that for the system captured in Fig 1-2A:

$$s_{1,2} = -(0.053)(4.42) \pm j\omega_n\sqrt{1-\zeta^2}$$

$$s_{1,2} = -0.23 \pm j4.42\sqrt{0.997}$$

$$s_{1,2} = -0.23 \pm 4.41j \tag{6}$$

From (6) one sees the real part of the complex roots is negative. Thus (5A) yields an exponentially decaying response; small values mean long settling times. The amplitude of oscillation is governed by imaginary values of the complex roots.

Exercise

1.2 Capture the time response of your motorized damped compound pendulum. Capture your plot (similar to Fig 1-2A left).

1.3 From your plot, identify the height of consequent peaks, and the time between peaks. Use these to calculate the damping ratio ζ and natural frequency ω_n for your system.

1.4 From your damping ratio and natural frequency, calculate the characteristic equation i.e. (4) and the roots i.e. (6)

Concept 2 System Modeling (Simulation)

Experimentally, the system's dynamics were captured and identified; Fig 1-2A and the equations above served to reveal the system's inherent (i.e. natural) characteristics. Equations (3A) and (3B) give us the pendulum's damping ratio and natural frequency respectively. One can now build a model to do a "sanity check" of the system that was identified.

There are many simulation tools on the market. *Simulink* (which is a *Matlab* toolbox) is a commercial package. *Xcos* (which is included in *Scilab*) is an open-source version.

Recall from (1) and (4), one has:

$$\ddot{\theta} = -2\zeta\omega_n\dot{\theta} - \omega_n^2 \tag{7}$$

$$\ddot{\theta} = -0.47\dot{\theta} - 19.5 \tag{8}$$

In Simulink (or Xcos) one can generate the following model:

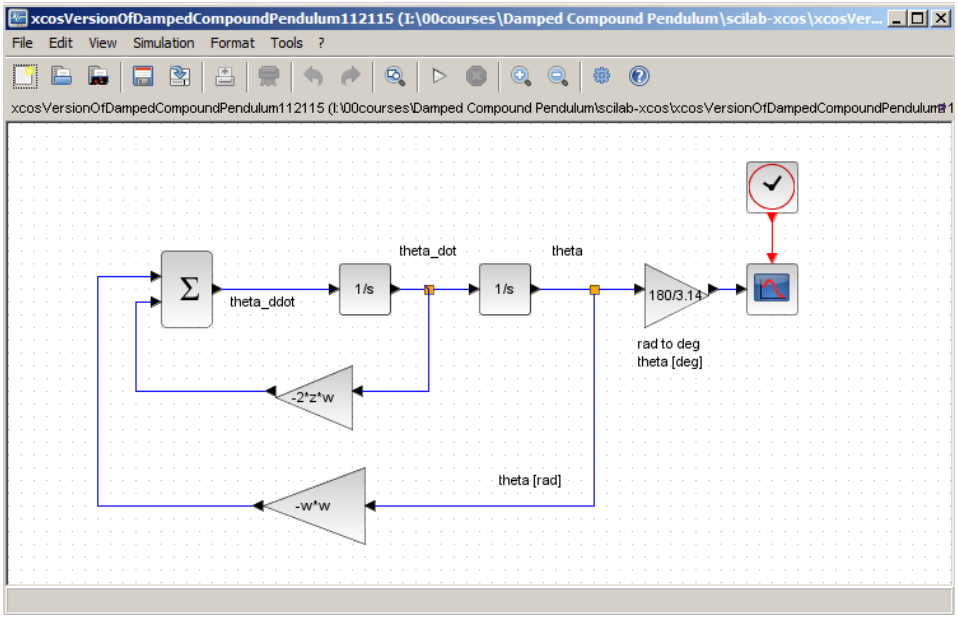


Fig 2-1A: Xcos implementation of damped compound pendulum Equation (7)

Executing the simulation of the model, one sees the damped compound pendulum's time response e.g. **Fig 2-1B**.

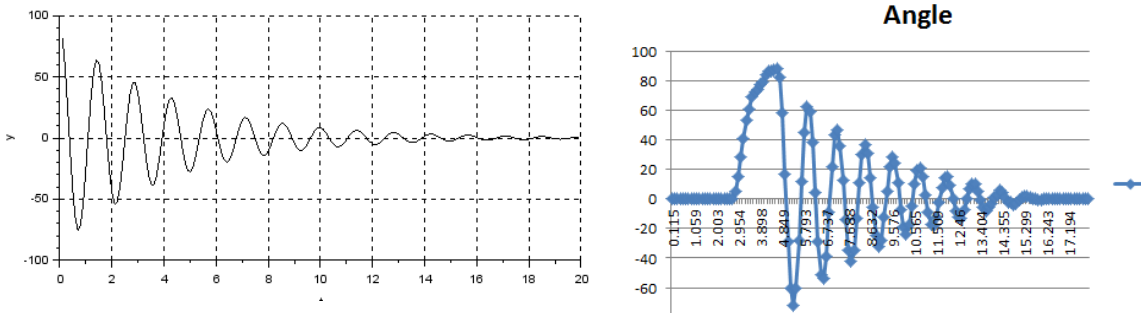


Fig 2-1B: Xcos simulation of pendulum time response (left). Contrast to experimentally captured response from Fig 1-2A shown again (right). Model appears to match experimental system closely

Exercise

2.1 Implement your model (i.e. damping ratio, natural frequency and initial angle) in Simulink (or Xcos) i.e. Replicate your own version of Fig 2-1A.

2.2 Execute the simulation of 2.1. Contrast the resulting plot (e.g. Fig 2-1B left) with your experimentally captured plot (e.g. Fig 2-1B right). From the simulation plot, show that the peak values and period are the same (or different) as your experimental plot.