Control System Design Project Proposal: The Magic Ball Balancing Beam

Alize Hall - 3323
Emily Williams - 3323
Patrick O'Keefe - 3323
Thomas Cunningham - 5182

MECA 482
Project Report
December 25, 2019

Department of Mechanical and Mechatronic Engineering
and Sustainable Manufacturing
California State University, Chico
Chico, CA 95929-078

1 – Introduction

The purpose of this project is to balance a ball bearing on a linear track, by controlling the incline angle of the track. The track inclination will be controlled by a servo motor connected to one end of the beam, while the center of the beam has a pin connection, allowing the beam to rotate in the x-y plane. The objective of this project is to create a mathematical model of the system, a control algorithm, and a simulation of the system in action.

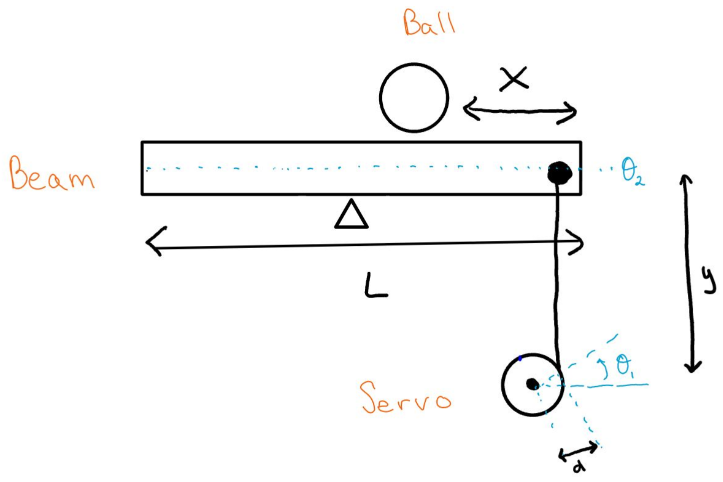


Figure 1: Physical schematic of desired project concept for ball & beam balancing model

2 - System Diagrams

The block diagram originally proposed at the beginning of this project can be seen in Figure 2 below. .The important pieces were an input/unit step of a desired position sent and summed with the output being the balls position. The controller then sends that information to the electro-mechanical ending in a new position location for the ball looping and starting the process over again.

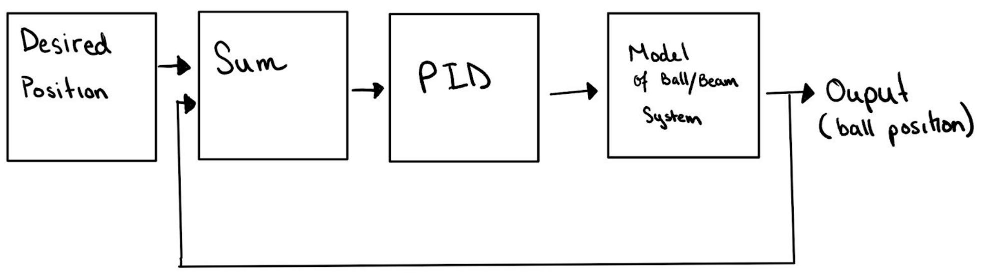


Figure 2: Proposed system layout for a ball and beam balancing model.

As a better understanding of how general control systems work, the system's block diagram evolved into the that shown in Figure 3. This schematic has more specific components and a more accurate representation of the desired model. The sensors, servo and controller were selected based on the block diagram and other requirements.

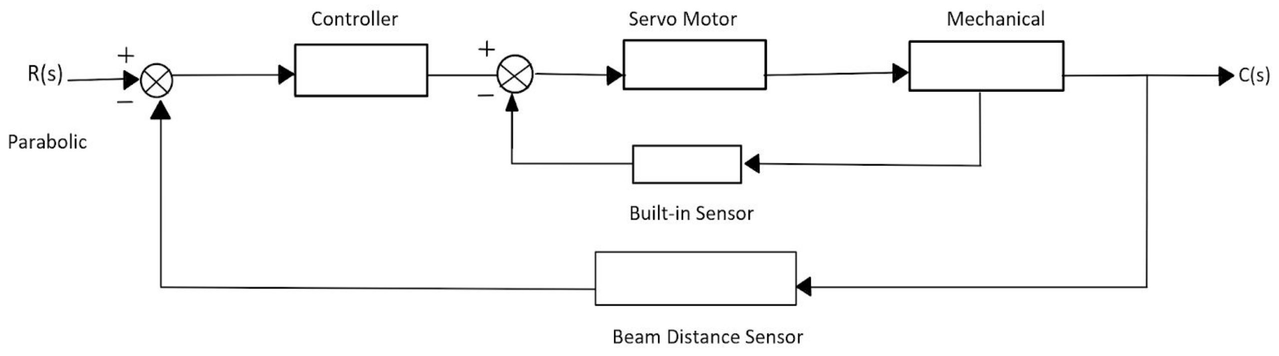


Figure 3: Block Diagram for ball balancing beam.

Our electromechanical diagram depicts the servo's output voltage, equivalent resistance and equivalent inductance which drives the rotational motion when adjusting the angle of the ball balance beam. The mechanical system is represented by the equivalent inertial mass, J_e. The output voltage of the servo applies a torque to its shaft, connected to the balance beam, and induces a torque and angular displacement on the mechanical system. See Figure 4 for a visual representation of this system. The transfer function and other supporting equations in this modeling of this system can be found in Appendix B and Appendix C.

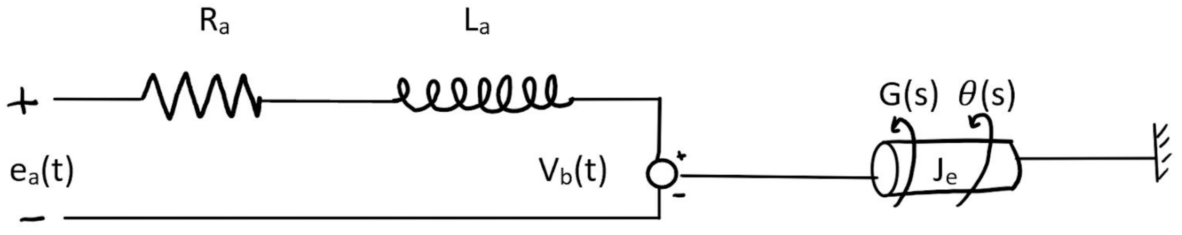


Figure 4: Mechanical Diagram for the system

3 - Sensor Calibration

Sensor calibration depends on what sensors and simulations are chosen for the system. The selected sensor chosen for this project was, an Ultrasonic Distance Sensor model HC-SR04 from Sparkfun. It has a measuring distance from 2cm to 400cm and an accuracy to 3 mm. The sensor will be hanging off at least 2cm from where the closest point the ball would roll in order to ensure the most accuracy. As the beam is only 50 cm long it is well within the range of the sensor. The cost was also an important factor when it came to the selection process, in the real world a budget would be given, the price of this sensor is $3.95. The VREP sensor is a simple ray proximity sensor with added values to model the Ultrasonic Distance Sensor. To calibrate the sensor it must first be sent a pulse, that triggers an input. That inputs then creates a range and will send a burst which changes it's "echo". The echo is an object that is a signal hitting an object within range disrupting it, and recording the distance where the location of the disturbance is happening (6).

4 - System Design and Simulations

The servo used to rotate the beam will be an RH-8D-3006 Mini 24V DC servo by Harmonic Drive Technologies. The total cost for this servo is servo's specifications are:

Table 1: Servo specifications (8)

Rated Power Output	6.2 W	**Gear Ratio**	1:100
Rated Voltage	24 V	**Maximum radial load**	196 N
Rated Current	0.8 A	**Maximum axial load**	98 N
Rated Output Torque	2 N-m	**Maximum Output Torque**	3.5 N-m
Rated Output Speed	30 rpm	**Maximum Output Speed**	50 rpm

A proportional-derivative controller was selected for this project as its primary use is to track a reference point in a simple system, i.e. the ball as the reference and the correcting beam being the system.

The Ziegler-Nichols method tunes the controller and can be proven done by hand or using an online tool. The selected controller was an Arduino Uno - R3, it’s a PID controller however we will set the I to zero, making it a PD controller since we do not need the Integral aspect for this project. The arduino is a great all encompassing USB compatible controller with a lot of research and resources making it the best option for the group. The price of the arduino is only $22.95 making it a cost effective and convenient controller.

5 - Matlab & Simulink

In order to determine the stability of the system and to determine the gain constants for a PD controller the system was modeled in Matlab. The model was comprised of physical constants and set limits as goals for the step response. The transfer function for the system was set up in the S-Domain. Also a PID model was created using gain constants as variables. These two models are then fed into a step response with the variable for step response to be plotted. From the plot we can determine that the gain constants need adjustment. We set up the plot to show horizontal lines for the +- 2% settling time, 5% overshoot limit, and the vertical line to show the goal of settling time. The last line of code spits out the resulting parameters of the step response. From the values we can determine if changes made to the gain constants help us achieve our goals and we can keep repeating this process to arrive to constants that optimize the system.

For the Simulink model a block diagram was setup for the plant of the system as well as for the whole system including a step input. For the plant of the system model a function was set up which represents the transfer function of the system relating the position of the ball to the angle of the beam. This sub system is then planted into an open loop with a step response to see the response of the system. The system is at first unstable meaning the ball runs away from the beam position, but a lead compensator with gain is added to compensate for the position of the roots. This stabilizes the response of the system. The simulation code can be found in Appendix A.

6 - Calculations

Through the use of Matlab, the systems servo would be tuned by adjusting the gain constant for the desired percent overshoot, rise time, settling time, and the peak time needed for the controls system. Shown below in Table 2 are a select few of the test parameters run through the model. The test runs selected are meant to demonstrate the relationship between the gain constants and the step response of the model. The goal was to get a settling time of 2.5 seconds. The changes to the values of Kp and Kd result in changes in the settling time, rise time and percent overshoot. When these gain constants are increased, the corresponding step parameters all decrease. However changing the step input affects the peak time; increasing the step input increases the peak time. The equations used for system tuning can be found in Appendix C.

Table 2: System Tuning

Step (m)	Kp (N-m/rad)	Kd (N-s/rad-s)	Settling Time	Rise Time	Peak Time	%OS
0.25	10	10	2.6479	0.3519	0.2827	13.0932
0.25	15	25	1.9446	0.1818	0.2611	4.4342
0.25	15	50	0.1638	0.1004	0.2532	1.2968
0.25	13	57	0.1491	0.0893	0.2520	0.8118
0.5	13	57	0.1491	0.0893	0.5041	0.8118

7 - V-rep Simulation

To validate the transfer function from Matlab, and the model developed in Simulink, the system was built and simulated in V-Rep. The beam and servo system were simplified so that the servo directly adjusts the angle of the beam with a 1:1 rotation. An ultrasonic sensor is attached to the end of the beam to measure the distance to the ball. V-REP outputs the distance value to Matlab to be the input value for simulink. We were unable to get the input signal to work correctly with matlab/Simulink, and thus were unable to run the simulation of the system.

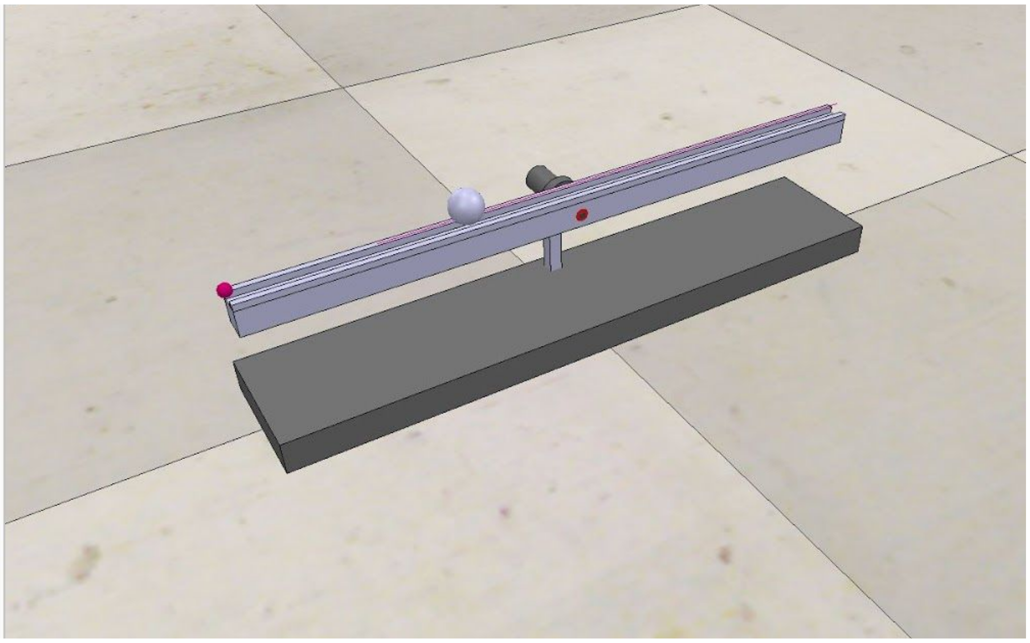


Figure 5: Ball balance beam V-Rep model

8 - Conclusion

The simulation used to model the ball balance beam system consists of a ball bearing and linear track, both controlled by a servo and PD controller to adjust the incline angle of the track. A mathematical model of the system was created in Matlab and adjusted by analysis through system tuning and step response plots. Simulink was utilized to create a control algorithm that would link the mathematical model in Matlab to control a virtual model created in V-Rep. Due to communication errors, the V-rep model was unable to process the input signal being sent from Simulink.

References

(1) Mechatronic Tutorials, Balancing of a Ball on Beam using Arduino as a PID controller, Retrieved by Oct. 17, 2019, from http://mechatronicstutorials.blogspot.com/2014/07/balancing-of-ball-on-beam-using-arduino.html

(2) Instructables, Ball Balancing Beam, Retrieved by Oct. 17, 2019, from https://www.instructables.com/id/Ball-Balancing-Beam/

(3) International Journal of Emerging Technology in Computer Science & Electronics (IJETCSE), MATHEMATICAL MODELLING OF BALL ON A MIDDLE SUPPORTED BEAM, Retrieved by Dec. 10, 2019 from http://www.ijetcse.com/wp-content/plugins/ijetcse/file/upload/docx/879MATHEMATICAL-MODELLING-OF-BALL-ON-A-MIDDLE-SUPPORTED-BEAM-pdf.pdf

(4) Ali, Awadalla & Taha, Osama & Naseraldeen, A & Ali, Taifour. (2017). Design and Implementation of Ball and Beam System Using PID Controller. 3. 1-4. 10.12691/acis-3-1-1, Retrieved Dec 10, 2019 from https://www.researchgate.net/publication/317079231_Design_and_Implementation_of_Ball_and_Beam_System_Using_PID_Controller

(5) Corel Office Document, Servo Tuning Principles, Retrieved Dec 10, 2019 from https://www.cnczone.com/forums/attachments/2/3/8/9/47749.attach

(6) ElecFreaks, "Ultrasonic Ranging Module HC - SR04 ." *HC-SR04*, Retrieved Dec 19 from https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

(7) Wang, Wei. "Control of a Ball and Beam System." The University of Adelaide, 5 June 2007.

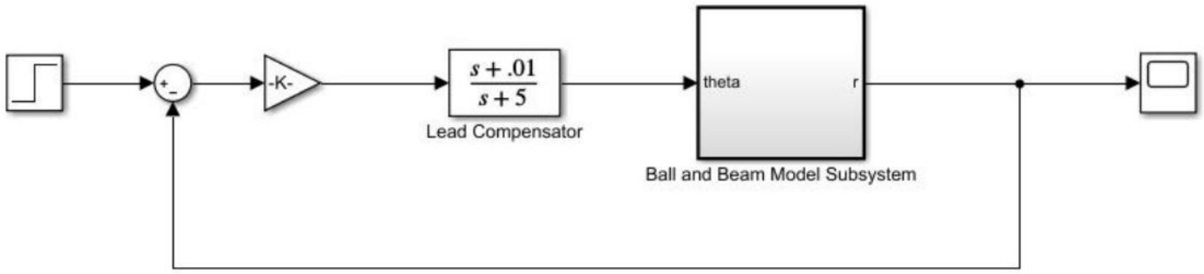
(8) "DC Servo System Catalog: RH Mini Series." Harmonic Drive Technologies. https://www.harmonicdrive.net/_hd/content/catalogs/pdf/dc_servo_catalog.pdf

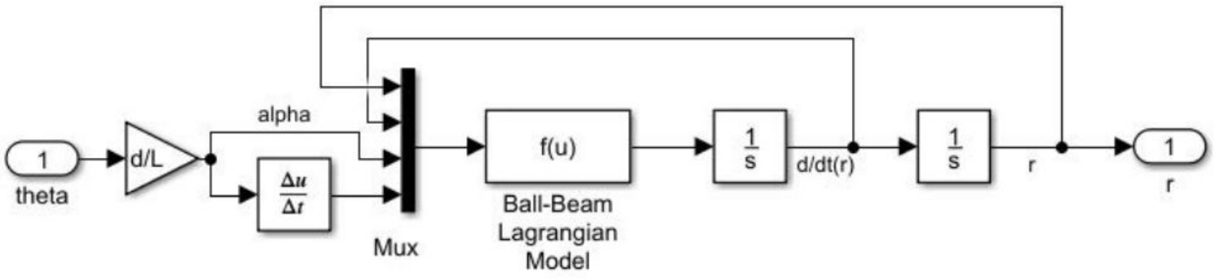
Appendix A: Mathematical Model Code

```
%% MECA 482 Group ID9
%% Team Members

"Patrick OKeefe";

"Alize Hall";

"Thomas Cunningham";

"Emily Williams";

%% Ball Balancing Beam Constants

clc; clear;

L = 1; %length of the beam (m)

d = 0.03; %servo arm length (m)

m = 0.065; %mass of the ball (kg)

r = 0.0127; %radius of the ball (m)

g = -9.8; %gravitational acceleration (m/s^2)

J = 2/5*m*(r^2); %balls moment of inertia (kg*m^2)

OS=0.05; %Percent overshoot as a percent of 1

Ts=1; %Settling time goal

S = tf('s'); %Transfer function in S-Domain

Pos_ball = -m*g*d/L/(J/r^2+m)/S^2 %Plant of the system P(s)

%% Root Locus for the system

rlocus(Pos_ball)

sgrid(0.7, 1.9)

axis([-5 5 -2 2])

Zo=0.01;

Po=5;

C=tf([1 Zo],[1,Po]);
```

```
rlocus(C*Pos_ball)

sgrid(0.7,1.9)

[k,poles]=rlocfind(C*Pos_ball)

%% Plotting Root Locus

sys_cl=feedback(k*C*Pos_ball,1);

t=0:0.01:5;

figure(2)

step(1*sys_cl,t)

%% Step Response for a PD controller

input_step=1; %step for the step response

Kp = 13; %Proportional gain

Kd = 57; %Derivative gain

Ki = 0; %Integral gain (zero because its not needed for this system)

C = pid(Kp,Ki,Kd); %function for PID controller

system=feedback(C*Pos_ball,1);

figure(1);

system_step=input_step*system;

[y]=[(1+OS)*input_step;0.98*input_step;1.02*input_step]; %setting up the horizontal lines for

percent OS and settling time

t=0:0.01:10;

step(system_step) %plots the step response for the system

line([0,t(end)],[y(1),y(1)],'Color','red'); %Percent Overshoot line, DO NOT CROSS

line([0,t(end)],[y(2),y(2)],'Color','green'); %98 Percent line for settling time

line([0,t(end)],[y(3),y(3)],'Color','green'); %102 Percent line for settling time
line([Ts,Ts],[0,10],'color','black','LineStyle',':') %Settling time goal

axis([0 1.25 0 (1.25*input_step)]); %sets up the axis limits for the plot

S=stepinfo(system_step) %system resultant parameters
```

Appendix B: Simulink





Appendix C: Equations used in mathematical

$$T(s)_{mech} = \frac{R(s)}{C(s)} = \frac{mgd}{L(\frac{J}{r^2} + m)} * \frac{1}{s^2} \qquad \left[\frac{m}{rad}\right]$$

$$C(s)_{servo} = \frac{k_d s^2 + K_p s + K_i}{s}$$

$$e(\infty) = \frac{1}{k_a} \qquad \begin{aligned} &k_a = Constant \\ &k_v = \infty \\ &kp = \infty \end{aligned}$$

$$Ts = \frac{4}{\zeta\omega_n} = \frac{4}{\sigma_d} \qquad Tp = \frac{\pi}{w_n\sqrt{1-\zeta^2}} = \frac{\pi}{\omega_d} \qquad \%OS = 100e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}$$

Appendix D: V-REP Script

```
-- This is the main script. The main script is not supposed to be modified,
-- unless there is a very good reason to do it.
-- Without main script,
-- there is no real simulation (child scripts are not called either in that case).
-- A main script marked as "default" (this is the default case) will use the
-- content of following file: system/dltmscpt.txt. This allows your old simulation
-- scenes to be automatically also using newer features, without explicitely coding
-- them. If you modify the main script, it will be marked as "customized", and you
-- won't benefit of that automatic forward compatibility mechanism.
simRemoteApi.start(19999)
function sysCall_init()
    sim.handleSimulationStart()
    sim.openModule(sim.handle_all)
    sim.handleGraph(sim.handle_all_except_explicit,0)
end


function sysCall_actuation()
    sim.resumeThreads(sim.scriptthreadresume_default)
    sim.resumeThreads(sim.scriptthreadresume_actuation_first)
    sim.launchThreadedChildScripts()
    sim.handleChildScripts(sim.syscb_actuation)
    sim.resumeThreads(sim.scriptthreadresume_actuation_last)
    sim.handleCustomizationScripts(sim.syscb_actuation)
    sim.handleAddOnScripts(sim.syscb_actuation)
    sim.handleSandboxScript(sim.syscb_actuation)
    sim.handleModule(sim.handle_all,false)
    simHandleJoint(sim.handle_all_except_explicit,sim.getSimulationTimeStep()) -- DEPRECATED
    simHandlePath(sim.handle_all_except_explicit,sim.getSimulationTimeStep()) -- DEPRECATED
    sim.handleMechanism(sim.handle_all_except_explicit)
    sim.handleIkGroup(sim.handle_all_except_explicit)
    sim.handleDynamics(sim.getSimulationTimeStep())
    sim.handleMill(sim.handle_all_except_explicit)
end


function sysCall_sensing()
    -- put your sensing code here
    sim.handleSensingStart()
    sim.handleCollision(sim.handle_all_except_explicit)
    sim.handleDistance(sim.handle_all_except_explicit)
    sim.handleProximitySensor(sim.handle_all_except_explicit)
    sim.handleVisionSensor(sim.handle_all_except_explicit)
    sim.resumeThreads(sim.scriptthreadresume_sensing_first)
    sim.handleChildScripts(sim.syscb_sensing)
    sim.resumeThreads(sim.scriptthreadresume_sensing_last)
    sim.handleCustomizationScripts(sim.syscb_sensing)
    sim.handleAddOnScripts(sim.syscb_sensing)
    sim.handleSandboxScript(sim.syscb_sensing)
    sim.handleModule(sim.handle_all,true)
    sim.resumeThreads(sim.scriptthreadresume_allnotyetresumed)


sim.handleGraph(sim.handle_all_except_explicit,sim.getSimulationTime()+sim.getSimulationTimeStep()
)
end


function sysCall_cleanup()
    sim.resetMilling(sim.handle_all)
    sim.resetMill(sim.handle_all_except_explicit)
    sim.resetCollision(sim.handle_all_except_explicit)
    sim.resetDistance(sim.handle_all_except_explicit)
    sim.resetProximitySensor(sim.handle_all_except_explicit)
    sim.resetVisionSensor(sim.handle_all_except_explicit)
    sim.closeModule(sim.handle_all)
end


function sysCall_suspend()
    sim.handleChildScripts(sim.syscb_suspend)
    sim.handleCustomizationScripts(sim.syscb_suspend)
    sim.handleAddOnScripts(sim.syscb_suspend)
    sim.handleSandboxScript(sim.syscb_suspend)
end


function sysCall_suspended()
```

```
    sim.handleCustomizationScripts(sim.syscb_suspended)
    sim.handleAddOnScripts(sim.syscb_suspended)
    sim.handleSandboxScript(sim.syscb_suspended)
end

function sysCall_resume()
    sim.handleChildScripts(sim.syscb_resume)
    sim.handleCustomizationScripts(sim.syscb_resume)
    sim.handleAddOnScripts(sim.syscb_resume)
    sim.handleSandboxScript(sim.syscb_resume)
end

-- By default threaded child scripts switch back to the main thread after 2 ms. The main
-- thread switches back to a threaded child script at one of above's "sim.resumeThreads"
-- location

-- You can define additional system calls here:
--[[
function sysCall_beforeCopy(inData)
    for key,value in pairs(inData.objectHandles) do
        print("Object with handle "..key.." will be copied")
    end
end
function sysCall_afterCopy(inData)
  for key,value in pairs(inData.objectHandles) do
    print("Object with handle "..key.." was copied")
  end
end


function sysCall_beforeDelete(inData)
  for key,value in pairs(inData.objectHandles) do
    print("Object with handle "..key.." will be deleted")
  end
  -- inData.allObjects indicates if all objects in the scene will be deleted
end


function sysCall_afterDelete(inData)
  for key,value in pairs(inData.objectHandles) do
    print("Object with handle "..key.." was deleted")
  end
  -- inData.allObjects indicates if all objects in the scene were deleted
end
--]]
```