The following code should be applied to the joints in the PID controller problems. The target position can then be specified in the dynamic properties of the joint.

function sysCall\_init()

-- do some initialization here

graph = sim.getObjectHandle('Graph')

PIDHandle = sim.getObjectHandle('Revolute\_joint1\_PID')

PIDjoint = sim.addGraphStream(graph,'PID Joint Velocity','deg/s',0,{1,0,0})

end

function sysCall\_actuation()

-- put your actuation code here

end

function sysCall\_jointCallback(inData)

-- put your actuation code here

-- This function gets called often, so it might slow down the simulation

-- (this is called at each dynamic simulation step, by default 10x more often than a child script)

-- We have:

-- inData.first : whether this is the first call from the physics engine, since the joint

-- was initialized (or re-initialized) in it.

-- inData.revolute : whether the joint associated with this script is revolute or prismatic

-- inData.cyclic : whether the joint associated with this script is cyclic or not

-- inData.handle : the handle of the joint associated with this script

-- inData.lowLimit : the lower limit of the joint associated with this script (if the joint is not cyclic)

-- inData.highLimit : the higher limit of the joint associated with this script (if the joint is not cyclic)

-- inData.passCnt : the current dynamics calculation pass. 1-10 by default. See next item for details.

-- inData.totalPasses : the number of dynamics calculation passes for each "regular" simulation pass.

-- 10 by default (i.e. 10\*5ms=50ms which is the default simulation time step)

-- inData.currentPos : the current position of the joint

-- inData.targetPos : the desired position of the joint

-- inData.errorValue : targetPos-currentPos (with revolute cyclic joints we take the shortest cyclic distance)

-- inData.effort : the last force or torque that acted on this joint along/around its axis. With Bullet,

-- torques from joint limits are not taken into account

-- inData.dynStepSize : the step size used for the dynamics calculations (by default 5ms)

-- inData.targetVel : the joint target velocity (as set in the user interface)

-- inData.maxForce : the joint maximum force/torque (as set in the user interface)

-- inData.velUpperLimit : the joint velocity upper limit (as set in the user interface)

--

-- Make sure that the joint is dynamically enabled, is in force/torque mode, motor enabled and

-- control loop enabled, otherwise this function won't be called

if inData.first then

PID\_P=0.1

PID\_I=0.1

PID\_D=0.1

pidCumulativeErrorForIntegralParam=0

end

-- The control happens here:

-- 1. Proportional part:

local ctrl=inData.errorValue\*PID\_P

-- 2. Integral part:

if PID\_I~=0 then

pidCumulativeErrorForIntegralParam=pidCumulativeErrorForIntegralParam+inData.errorValue\*inData.dynStepSize

else

pidCumulativeErrorForIntegralParam=0

end

ctrl=ctrl+pidCumulativeErrorForIntegralParam\*PID\_I

-- 3. Derivative part:

if not inData.first then

ctrl=ctrl+(inData.errorValue-pidLastErrorForDerivativeParam)\*PID\_D/inData.dynStepSize

end

pidLastErrorForDerivativeParam=inData.errorValue

-- 4. Calculate the velocity needed to reach the position in one dynamic time step:

local maxVelocity=ctrl/inData.dynStepSize -- max. velocity allowed.

if (maxVelocity>inData.velUpperLimit) then

maxVelocity=inData.velUpperLimit

end

if (maxVelocity<-inData.velUpperLimit) then

maxVelocity=-inData.velUpperLimit

end

local forceOrTorqueToApply=inData.maxForce -- the maximum force/torque that the joint will be able to exert

-- 5. Following data must be returned to CoppeliaSim:

firstPass=false

local outData={}

outData.velocity=maxVelocity

outData.force=forceOrTorqueToApply

return outData

end

function sysCall\_sensing()

-- put your sensing code here

sim.setGraphStreamValue(graph,PIDjoint,180\*sim.getJointVelocity(PIDHandle)/math.pi)

end

function sysCall\_cleanup()

-- do some clean-up here

end

-- See the user manual or the available code snippets for additional callback functions and details

The following code should be applied to the joints in the Inverse Dynamics problems. The target position can then be specified in the dynamic properties of the joint.

function sysCall\_init()

-- do some initialization here

graph = sim.getObjectHandle('Graph')

InverseHandle = sim.getObjectHandle('Revolute\_joint1\_Inverse')

Inversejoint = sim.addGraphStream(graph,'Inverse Joint Velocity','deg/s',0,{0,1,0})

end

function sysCall\_actuation()

-- put your actuation code here

end

function sysCall\_jointCallback(inData)

-- put your actuation code here

-- This function gets called often, so it might slow down the simulation

-- (this is called at each dynamic simulation step, by default 10x more often than a child script)

-- We have:

-- inData.first : whether this is the first call from the physics engine, since the joint

-- was initialized (or re-initialized) in it.

-- inData.revolute : whether the joint associated with this script is revolute or prismatic

-- inData.cyclic : whether the joint associated with this script is cyclic or not

-- inData.handle : the handle of the joint associated with this script

-- inData.lowLimit : the lower limit of the joint associated with this script (if the joint is not cyclic)

-- inData.highLimit : the higher limit of the joint associated with this script (if the joint is not cyclic)

-- inData.passCnt : the current dynamics calculation pass. 1-10 by default. See next item for details.

-- inData.totalPasses : the number of dynamics calculation passes for each "regular" simulation pass.

-- 10 by default (i.e. 10\*5ms=50ms which is the default simulation time step)

-- inData.currentPos : the current position of the joint

-- inData.targetPos : the desired position of the joint

-- inData.errorValue : targetPos-currentPos (with revolute cyclic joints we take the shortest cyclic distance)

-- inData.effort : the last force or torque that acted on this joint along/around its axis. With Bullet,

-- torques from joint limits are not taken into account

-- inData.dynStepSize : the step size used for the dynamics calculations (by default 5ms)

-- inData.targetVel : the joint target velocity (as set in the user interface)

-- inData.maxForce : the joint maximum force/torque (as set in the user interface)

-- inData.velUpperLimit : the joint velocity upper limit (as set in the user interface)

--

-- Make sure that the joint is dynamically enabled, is in force/torque mode, motor enabled and

-- control loop enabled, otherwise this function won't be called

if inData.first then

PID\_P=0.1

PID\_I=0.1

PID\_D=0.1

pidCumulativeErrorForIntegralParam=0

end

-- The control happens here:

-- 1. Proportional part:

local ctrl=inData.errorValue\*PID\_P

-- 2. Integral part:

if PID\_I~=0 then

pidCumulativeErrorForIntegralParam=pidCumulativeErrorForIntegralParam+inData.errorValue\*inData.dynStepSize

else

pidCumulativeErrorForIntegralParam=0

end

ctrl=ctrl+pidCumulativeErrorForIntegralParam\*PID\_I

-- 3. Derivative part:

if not inData.first then

ctrl=ctrl+(inData.errorValue-pidLastErrorForDerivativeParam)\*PID\_D/inData.dynStepSize

end

pidLastErrorForDerivativeParam=inData.errorValue

-- 4. Inverse Dyanmics Part:

M = 1

g = 9.81

l = 0.25

I = 0.003083

H = I

C = 0

tau = M\*g\*l\*math.sin(inData.currentPos)

q\_d = inData.errorValue/inData.dynStepSize

if not inData.first then

qq\_d = ((inData.targetVel - ((inData.errorValue-pidLastErrorForDerivativeParam)/inData.dynStepSize)) - passLastAccelError)/inData.dynStepSize

ctrl=H\*(ctrl + qq\_d) + C\*q\_d

end

pidLastErrorForDerivativeParam=inData.errorValue

passLastAccelError = inData.targetVel - ((inData.errorValue-pidLastErrorForDerivativeParam)/inData.dynStepSize)

-- 4. Calculate the velocity needed to reach the position in one dynamic time step:

local maxVelocity=ctrl/inData.dynStepSize -- max. velocity allowed.

if (maxVelocity>inData.velUpperLimit) then

maxVelocity=inData.velUpperLimit

end

if (maxVelocity<-inData.velUpperLimit) then

maxVelocity=-inData.velUpperLimit

end

local forceOrTorqueToApply=inData.maxForce -- the maximum force/torque that the joint will be able to exert

-- 5. Following data must be returned to CoppeliaSim:

firstPass=false

local outData={}

outData.velocity=maxVelocity

outData.force=forceOrTorqueToApply

return outData

end

function sysCall\_sensing()

-- put your sensing code here

sim.setGraphStreamValue(graph,Inversejoint,180\*sim.getJointVelocity(InverseHandle)/math.pi)

end

function sysCall\_cleanup()

-- do some clean-up here

end

-- See the user manual or the available code snippets for additional callback functions and details