```
require('RigidBodyWin/subRoutines/ConvexHull2D')
require("module")
HessianQuadratic.add=QuadraticFunctionHardCon.add

-- ( x_i -value)^2
function HessianQuadratic:addD(index, value)
 local i=CT.ivec(index)
 local v=CT.vec(1,-1*value)
 self:addSquared(i,v)
end


-- PDServo class
--class 'QPservo'
QPservo=LUAclass()

function QPservo:setCoef(dofInfo,kp, kd, tgtVelScale,
k_scale)
 kp:setSize(dofInfo:numDOF())
 kp:setAllValue(k_p)
 kd:setSize(dofInfo:numDOF())
 kd:setAllValue(k_d)
 tgtVelScale:setSize(dofInfo:numDOF())
 tgtVelScale:setAllValue(k_d)

 -- exclude root joint
 kp:range(0,7):setAllValue(0)
 kd:range(0,7):setAllValue(0)
 tgtVelScale:range(0,7):setAllValue(0)

 print("initQPservo:"..dofInfo:skeleton():bone(1):name())
 for i=2,dofInfo:skeleton():numBone()-1 do
  local bone=dofInfo:skeleton():bone(i)
  local vbone=bone:treeIndex()
  local nJoint=dofInfo:numDOF(vbone)
  --      print("initQPservo:"..bone:name())
  for j=0, nJoint-1 do

   local dofIndex=dofInfo:DOFindex(vbone,j)

   kp:set(dofIndex, k_p*k_scale.default[1])
   kd:set(dofIndex, k_d*k_scale.default[2])
   tgtVelScale:set(dofIndex, k_scale.default[3])

   if bone:voca()==MotionLoader.LEFTANKLE or bone:voca
()==MotionLoader.RIGHTANKLE then
    if k_scale.ankle~=nil then
     kp:set(dofIndex, k_p*k_scale.ankle[1])
     kd:set(dofIndex, k_d*k_scale.ankle[2])
     tgtVelScale:set(dofIndex, k_scale.ankle[3])
    end
   elseif bone:voca()==MotionLoader.LEFTCOLLAR or bone:voca
()==MotionLoader.RIGHTCOLLAR then
    if k_scale.collar~=nil then
     kp:set(dofIndex, k_p*k_scale.collar[1])
     kd:set(dofIndex, k_d*k_scale.collar[2])
     tgtVelScale:set(dofIndex, k_scale.collar[3])
    end
   elseif bone:voca()==MotionLoader.LEFTSHOULDER or
bone:voca()==MotionLoader.RIGHTSHOULDER then
    if k_scale.shoulder~=nil then
     kp:set(dofIndex, k_p*k_scale.shoulder[1])
     kd:set(dofIndex, k_d*k_scale.shoulder[2])
     tgtVelScale:set(dofIndex, k_scale.shoulder[3])
    end
   elseif bone:voca()==MotionLoader.LEFTELBOW or bone:voca
()==MotionLoader.RIGHTELBOW then
    if k_scale.elbow~=nil then
     kp:set(dofIndex, k_p*k_scale.elbow[1])
     kd:set(dofIndex, k_d*k_scale.elbow[2])
     tgtVelScale:set(dofIndex, k_scale.elbow[3])
    end
   elseif bone:voca()==MotionLoader.LEFTKNEE or bone:voca
()==MotionLoader.RIGHTKNEE then
    if k_scale.knee~=nil then
     kp:set(dofIndex, k_p*k_scale.knee[1])
     kd:set(dofIndex, k_d*k_scale.knee[2])
     tgtVelScale:set(dofIndex, k_scale.knee[3])
    end
   elseif bone:voca()==MotionLoader.LEFTHIP or bone:voca
()==MotionLoader.RIGHTHIP then
    if k_scale.hip~=nil then
     kp:set(dofIndex, k_p*k_scale.hip[1])
     kd:set(dofIndex, k_d*k_scale.hip[2])
```

```
  tgtVelScale:set(dofIndex, k_scale.hip[3])
 end
elseif bone:voca()==MotionLoader.CHEST then
 if k_scale.chest~=nil then
  kp:set(dofIndex, k_p*k_scale.chest[1])
  kd:set(dofIndex, k_d*k_scale.chest[2])
  tgtVelScale:set(dofIndex, k_scale.chest[3])
 end
elseif bone:voca()==MotionLoader.CHEST2 then
 if k_scale.chest2~=nil then
  kp:set(dofIndex, k_p*k_scale.chest2[1])
  kd:set(dofIndex, k_d*k_scale.chest2[2])
  tgtVelScale:set(dofIndex, k_scale.chest2[3])
 end
elseif bone:voca()==MotionLoader.NECK then
 if k_scale.neck~=nil then
  kp:set(dofIndex, k_p*k_scale.neck[1])
  kd:set(dofIndex, k_d*k_scale.neck[2])
  tgtVelScale:set(dofIndex, k_scale.neck[3])
 end
elseif bone:voca()==MotionLoader.HEAD then
 if k_scale.head~=nil then
  kp:set(dofIndex, k_p*k_scale.head[1])
  kd:set(dofIndex, k_d*k_scale.head[2])
  tgtVelScale:set(dofIndex, k_scale.head[3])
 end
end
if str_include(bone:name(), "toes") then
 local dofIndex=dofInfo:DOFindex(vbone,j)
 if k_scale.toes~=nil then
  kp:set(dofIndex, k_p*k_scale.toes[1])
  kd:set(dofIndex, k_d*k_scale.toes[2])
  tgtVelScale:set(dofIndex, k_scale.toes[3])
 end

end

if dofInfo:DOFtype(vbone, j)==MotionDOFinfo.SLIDE then
 local dofIndex=dofInfo:DOFindex(vbone,j)
 kp:set(dofIndex, k_p_slide)
 kd:set(dofIndex, k_d_slide)
 tgtVelScale:set(dofIndex, 0)
```

```
   end
  end
 end
end

function QPservo:updateCoef()
 local dofInfo=self.dofInfo

 k_p=model.k_p_ID or 100 -- hyunwoo 100
 k_d=model.k_d_ID or 30 -- hyunwoo 50
 print("K=", k_p, k_d)
 k_p_slide=model.k_p_ID*5
 k_d_slide=model.k_d_ID*5

 -- self:setIDGain(dofInfo:skeleton(), self.kp_id,
self.kd_id, k_p, k_d, k_p_slide or k_p*5, k_d_slide or
k_d*5)
 local unused=vectorn()
 self:setCoef(dofInfo, self.kp_id, self.kd_id, unused,
model.k_scale_id)

 k_p=model.k_p_PD or 100 -- hyunwoo 100
 k_d=model.k_d_PD or 10 -- hyunwoo 50
 print("K_pd=", k_p, k_d)
 k_p_slide=model.k_p_PD*100
 k_d_slide=model.k_d_PD*100

 local k_scale_active=model.k_scale_active_pd

 self:setCoef(dofInfo,self.kp_active, self.kd_active,
self.tgtVelScale_active, k_scale_active)

 local k_scale_passive=model.k_scale_passive_pd

 self:setCoef(dofInfo,self.kp_passive, self.kd_passive,
self.tgtVelScale_passive, k_scale_passive)
end

function QPservo:__init(dofInfo,timestep,integrator)
 self.state={previousFlightPhase=false, flightPhase=false,
supportPhaseElapsed=100, flightPhaseElapsed=0}
 self.theta=vectorn()
 self.dtheta=vectorn()
 -- HD servo
 self.theta_d=vectorn() -- desired q
 self.dtheta_d=vectorn() -- desired dq
 self.ddtheta_d=vectorn() -- desired ddq

 -- PD servo
 self.theta_d_pd=vectorn()

 self.desiredacceleration=vectorn()
 self.controlforce=vectorn()
 self.kp=vectorn()
 self.kd=vectorn()
 self.kp_id=vectorn()
 self.kd_id=vectorn()

 self.tgtVelScale=vectorn()
 self.kp_active=vectorn()
 self.kd_active=vectorn()
 self.tgtVelScale_active=vectorn()
 self.kp_passive=vectorn()
 self.kd_passive=vectorn()
 self.tgtVelScale_passive=vectorn()
 self.mask_slide=vectorn()


 -- lleg+rleg+upperbody=all
 self.mask_lleg=vectorn() -- excluding sliding joints
 self.mask_rleg=vectorn() -- excluding sliding joints
 self.mask_upperbody=vectorn()
 self.scale_lleg=1
 self.scale_rleg=1
 self.scale_upperbody=1

 self.muscleActiveness=0.3
 self.kp_weight=1.0 -- use kp_active(1) or kp_passive(0)
 self.kd_weight=1.0 -- use kd_active(1) or kd_passive(0)
 self.mask_slide:setSize(dofInfo:numDOF())
 self.mask_slide:setAllValue(0)
 self.mask_lleg:setSize(dofInfo:numDOF())
 self.mask_rleg:setSize(dofInfo:numDOF())
 self.mask_upperbody:setSize(dofInfo:numDOF())
```

```
 self.mask_lleg:setAllValue(0)
 self.mask_rleg:setAllValue(0)
 self.mask_upperbody:setAllValue(1)

 self.dofInfo=dofInfo
 self:updateCoef()
 print ("kp=",self.kp)
 print ("kd=",self.kd)

 local skel=dofInfo:skeleton()

 local lhip=skel:getBoneByVoca(MotionLoader.LEFTHIP)
 local rhip=skel:getBoneByVoca(MotionLoader.RIGHTHIP)

 self.lkneeDOF=dofInfo:DOFindex(skel:getBoneByVoca
(MotionLoader.LEFTKNEE):treeIndex(),0)
 self.rkneeDOF=dofInfo:DOFindex(skel:getBoneByVoca
(MotionLoader.RIGHTKNEE):treeIndex(),0)
 local function setClampMax(clampForce, clampTorque)
  local clampMax=vectorn(dofInfo:numDOF())
  clampMax:setAllValue(0)
  for i=2,skel:numBone()-1 do
   local bone=skel:bone(i)
   local vbone=bone:treeIndex()
   local nJoint=dofInfo:numDOF(vbone)
   for j=0, nJoint-1 do
    local dofIndex=dofInfo:DOFindex(vbone,j)
    if dofInfo:DOFtype(vbone, j)==MotionDOFinfo.SLIDE then
     local dofIndex=dofInfo:DOFindex(vbone,j)
     self.mask_slide:set(dofIndex, 1)
     clampMax:set(dofIndex, clampForce)
    else
     clampMax:set(dofIndex, clampTorque)

     if bone:isDescendent(lhip) then
      self.mask_lleg:set(dofIndex,1)
      self.mask_upperbody:set(dofIndex,0)
     elseif bone:isDescendent(rhip) then
      self.mask_rleg:set(dofIndex,1)
      self.mask_upperbody:set(dofIndex,0)
     end
```

```lua
    end
   end
  end
  return clampMax
 end

 local clampTorque=model.clampTorqueID or 400
 local clampForce=model.clampForceID or 4000

 self.clampMaxID=setClampMax(clampForce, clampTorque)

 clampTorque=model.clampTorque or 800
 clampForce=model.clampForce or 8000

 self.clampMax=setClampMax(clampForce, clampTorque)

 self.clampMin=self.clampMax*-1
 self.clampMinID=self.clampMaxID*-1

 self.numActualDOF=dofInfo:numActualDOF()
 self.workspace={}
 local w=self.workspace
 w.M=matrixn()
 w.b=vectorn(self.numActualDOF)
 w.JtV=matrixn()
 w.Mlcp=matrixn()
 w.Mlcp_bias=vectorn()
 w.CE=matrixn()
 w.ce0=vectorn()
 w.CI=matrixn()
 w.ci0=vectorn()
 w.x=vectorn()

 return o
end

function QPservo:initQPservo(startf, endf,motionDOF,
dmotionDOF, ddmotionDOF, motionDOF_pdtarget)

 self.startFrame=startf
 self.endFrame=endf
 self.currFrame=startf
 self.deltaTime=0
 self.motionDOF=motionDOF
 self.dmotionDOF=dmotionDOF
 self.ddmotionDOF=ddmotionDOF
 self.motionDOF_pdtarget=motionDOF_pdtarget or motionDOF

end

-- generate FBtorque
function QPservo:generateTorque(simulator, maxForce)

 self.currFrame=(simulator:currentTime()
+self.deltaTime)*model.frame_rate+self.startFrame
 --print(self.currFrame) -- extremely slow.
 if self.currFrame>self.endFrame-1 then
  simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VALUE, self.theta)
  simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VELOCITY, self.dtheta)
  return false
 end

 self:_generateTorque(simulator, self.currFrame, maxForce)
 return true
end
function QPservo:_calcDesiredAcceleration()
 local state=self.theta
 local dstate=self.dtheta
 self.desiredacceleration:setSize(self.motionDOF:numDOF())

 --   self.desiredacceleration:assign(self.kp*
(self.theta_d-state)+
 --    self.kd*(self.dtheta_d-dstate))

 local delta=self.theta_d-state
 MainLib.VRMLloader.projectAngles(delta) -- [-pi, pi]


 self.desiredacceleration:assign(self.kp_id*delta +
self.kd_id*(self.dtheta_d*(useCase.QPservoDScaleCoef or
1.0)-dstate))
```

```
 self.desiredacceleration:smoothClamp(-400, 400)
 self.desiredacceleration:radd(self.ddtheta_d)
 --self.desiredacceleration:clamp(-400, 400)

end



-- deprecated: use _calcDesiredAcceleration
function QPservo:calcDesiredAcceleration(simulator, frame,
state, dstate)


 --[[ continuous sampling ]]--
 --   print("theta",self.theta)

 self:sampleTargetPoses(frame)

 --   self.dtheta_d:setAllValue(0)
 self:_calcDesiredAcceleration()
end

function QPservo:sampleCurrPose(simulator)
 simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VALUE, self.theta)
 simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VELOCITY, self.dtheta)
end
function QPservo:sampleTargetPoses( frame)
 -- desired (target) pose
 self.motionDOF:samplePose(frame, self.theta_d)
 self.motionDOF_pdtarget:samplePose(frame, self.theta_d_pd)
 self.dmotionDOF:sampleRow(frame, self.dtheta_d)
 MotionDOF.convertDPoseToDState(self.theta_d,self.dtheta_d)
 self.ddmotionDOF:sampleRow(frame, self.ddtheta_d)
 MotionDOF.convertDPoseToDState
(self.theta_d,self.ddtheta_d)

end

function QPservo:addPDtorque(simulator)
 -- pdservo
 if mrd_info and mrd_info.outputContactForce and
usePenaltyMethod then
  local cqInfo=simulator:queryContactAll()
  local collector=mrd_info.outputContactForce[2]
  for i=0, cqInfo:size()-1 do
   if cqInfo(i).chara==0 then
    local bone=cqInfo(i).bone
    if bone:name()=="lfoot" or bone:name()=="ltoes" then
     collector[1]:radd(simulator:getWorldState
(0):globalFrame(bone):toGlobalDir(cqInfo(i).f))
    else
     collector[2]:radd(simulator:getWorldState
(0):globalFrame(bone):toGlobalDir(cqInfo(i).f))
    end
   end
  end
 end
 self.kp:interpolate(self.kp_weight, self.kp_passive,
self.kp_active)
 self.kd:interpolate(self.kd_weight, self.kd_passive,
self.kd_active)
 self.tgtVelScale:interpolate(self.kd_weight,
self.tgtVelScale_passive, self.tgtVelScale_active)
 local delta_pd=self.theta_d_pd-self.theta
 MainLib.VRMLloader.projectAngles(delta_pd) -- [-pi, pi]

 local pdforce=self.kp*delta_pd + self.kd*
(self.dtheta_d*self.muscleActiveness*self.tgtVelScale-
self.dtheta)
 pdforce:clamp(self.clampMin, self.clampMax)
 ---- taesoo debug..
 --pdforce:range(0,7):setAllValue(0)
 --self.controlforce:range(0,7):setAllValue(0)

 self.controlforce:radd(pdforce)
 do return end

 if self.kneeTorqueL then
  self.controlforce:set(self.lkneeDOF, self.controlforce
(self.lkneeDOF)+self.kneeTorqueL)
  self.controlforce:set(self.rkneeDOF, self.controlforce
(self.rkneeDOF)+self.kneeTorqueR)
 end

 if g_debugOneStep and g_debugOneStepFlag then

  for i=0, 6 do
   assert(pdforce(i)==0)
  end
  g_debugOneStep:pushBack(saveDebugInfo(simulator))
  if simulator._debugInfo:length()~=0 then
   g_debugOneStep:pushBack(tostring(simulator._debugInfo))
   simulator._debugInfo:assign("")
  end
  g_debugOneStep:pushBack({"theta",self.theta:copy()})
  g_debugOneStep:pushBack({"dtheta",self.dtheta:copy()})
  g_debugOneStep:pushBack({"dtheta_d",self.dtheta_d:copy()}
)
  g_debugOneStep:pushBack
({"theta_d_pd",self.theta_d_pd:copy(),self.kp, self.kd,
self.tgtVelScale})
  g_debugOneStep:pushBack
({"desiredAcc",self.desiredacceleration:copy()})
  g_debugOneStep:pushBack
({"controlforce",self.controlforce:copy()})
  g_debugOneStep:pushBack({"pdforce",pdforce:copy()})
  local cqInfo=simulator:queryContactAll()
  g_debugOneStep:pushBack({"cqInfo", cqInfo:size()})
  for i=0, cqInfo:size()-1 do
   if cqInfo(i).chara==0 then
    g_debugOneStep:pushBack({cqInfo(i).bone:name(), cqInfo
(i).p:copy(), cqInfo(i).tau:copy(), cqInfo(i).f:copy(), })
   end
  end

  --g_debugOneStepFlag=false -- store all frames or just
the first simulation frames
 end

end
function QPservo:_generateTorque(simulator, frame,
maxForce, swingFoot)
```

```
 simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VALUE, self.theta)
 simulator:getLinkData(0,
Physics.DynamicsSimulator.JOINT_VELOCITY, self.dtheta)
 self:calcDesiredAcceleration(simulator, frame,
self.theta, self.dtheta)

 local link_pair_count=simulator:getNumContactLinkPairs()
 local numActualDOF=self.numActualDOF
 local w=self.workspace
 simulator:calcMassMatrix2(0, w.M, w.b)
 self.controlforce:setSize(numActualDOF+1)
 self.controlforce:setAllValue(0)
 if link_pair_count>0 then
  simulator:calcContactJacobian(w.JtV, link_pair_count)
  local cdim=w.JtV:cols()
  local totalDIM=numActualDOF*2+cdim -- ddq, tau, lambda
  local qp=HessianQuadratic(totalDIM)
  -- minimize desired acc error
  for i=6,numActualDOF-1 do
   qp:addD(i,self.desiredacceleration(i+1))
  end
  -- minimize joint torque
  for i=0,numActualDOF-1 do
   qp:addD(i+numActualDOF,0)
  end
  -- minimize contact force
  for i=0,cdim-1 do
   qp:addD(i+numActualDOF*2,0)
  end
  for i=0,6 do
   qp:addD(i,0)
  end
  -- set equality constraints
  w.CE:setSize(numActualDOF, totalDIM)
  w.CE:sub(0,numActualDOF,0,numActualDOF):assign(w.M)
  local minusI=w.CE:sub
(0,numActualDOF,numActualDOF,numActualDOF*2)
  minusI:identity()
  minusI:rmult(-1)
  local minusJtV=w.CE:sub(0,numActualDOF, numActualDOF*2,
totalDIM)
```

```
  minusJtV:assign(w.JtV)
  minusJtV:rmult(-1)
  w.ce0:assign(w.b)

  -- set inequality constraints
  simulator:getLCPmatrix(w.Mlcp, w.Mlcp_bias)
  assert(w.Mlcp_bias:size()==cdim)
  -- local t=vectorn()
  -- simulator:getLCPsolution(t)
  w.CI:setSize(w.Mlcp:rows()+cdim, totalDIM)
  w.CI:setAllValue(0)
  w.CI:sub(0, w.Mlcp:rows(), numActualDOF*2,
totalDIM):assign(w.Mlcp)
  w.CI:sub(w.Mlcp:rows(), 0, numActualDOF*2,
totalDIM):identity()
  w.ci0:setSize(w.Mlcp:rows()+cdim)
  w.ci0:setAllValue(0)
  w.ci0:range(0, w.Mlcp:rows()):assign(w.Mlcp_bias)
  qp:solveQuadprog(w.CE, w.ce0, w.CI, w.ci0, w.x)

  assert(w.x==w.x)
  self.controlforce:range(0,7):setAllValue(0)
  self.controlforce:range(7,self.controlforce:size
()):assign(w.x:range(numActualDOF+6,numActualDOF*2))
  print(self.controlforce)
  -- self:addPDtorque(simulator)
  dbg.console()

 else
  local totalDIM=numActualDOF*2 -- ddq and tau
  local qp=HessianQuadratic(totalDIM)
  -- dbg.console()
  for i=6,numActualDOF-1 do
   qp:addD(i,self.desiredacceleration(i+1))
  end
  for i=0,numActualDOF-1 do
   qp:addD(i+numActualDOF,0)
  end
  for i=0,6 do
   qp:addD(i,0)
  end
  w.CE:setSize(numActualDOF+6, numActualDOF*2)
  w.CE:sub(0,numActualDOF,0,numActualDOF):assign(w.M)
  local minusI=w.CE:sub
(0,numActualDOF,numActualDOF,numActualDOF*2)
  minusI:identity()
  minusI:rmult(-1)
  -- constrain tau[0:6]=0
  w.CE:sub(numActualDOF, numActualDOF+6):setAllValue(0)
  w.CE:sub(numActualDOF, numActualDOF+6, numActualDOF,
numActualDOF+6):identity()
  w.ce0:setSize(numActualDOF+6)
  w.ce0:range(0,numActualDOF):assign(w.b)
  w.ce0:range(numActualDOF,numActualDOF+6):setAllValue(0)
  w.CI:setSize(0,0)
  w.ci0:setSize(0)
  qp:solveQuadprog(w.CE, w.ce0, w.CI, w.ci0, w.x)
--  dbg.console()
  --self:addPDtorque(simulator)
  self.controlforce:range(0,7):setAllValue(0)
  self.controlforce:range(7,self.controlforce:size
()):assign(w.x:range(numActualDOF+6,numActualDOF*2))
 end
end

function QPservo:calcContactCentroid(simulator, graph,
swingFoot)
 local contactHull=self.contactHull
 assert(contactHull)
 --assert(contactHull.N>=1)
 if contactHull.N==0 then RE.output2
("warning","contactHull.N==0") end

 if swingFoot~="L" then
  local frameL=simulator:getWorldState(0):globalFrame
(graph.lfoot)
  -- it's safe to include the current heel and toe
positions in the support polygon.
  -- This allows faster swiching between heel and toe
supports.
  contactHull:addVector3(frameL.translation)
  contactHull:addVector3(frameL:toGlobalPos
(graph.lfootpos))
 end
```

```lua
 if swingFoot~="R" then
  local frameR=simulator:getWorldState(0):globalFrame
(graph.rfoot)
  contactHull:addVector3(frameR.translation)
  contactHull:addVector3(frameR:toGlobalPos
(graph.rfootpos))
 end

 contactHull:buildHull()

 local centroid, area=contactHull:calcCentroid()
 centroid=vector3(centroid.x, 0, centroid.y)
 return centroid
end

function QPservo:rewindTargetMotion(simulator)
 self.deltaTime=-1*simulator:currentTime()
end
```