Simulador VREP

SceneControl

```
-------------------------------------------------------------------------------------------------------------
-- Función para limpiar la escena y cargar un nuevo modelo
-------------------------------------------------------------------------------------------------------------
function recursiveRemove(parentNode)
    local childList = sim.getObjectsInTree(parentNode, sim.handle_all, 3)

    if #childList == 0 or childList == nil then
        sim.removeObject(parentNode)
    else
        for i=1,#childList,1 do
            recursiveRemove(childList[i])
        end
        sim.removeObject(parentNode)
    end
end

function clearScene(u, id)
    local objectList = sim.getObjectsInTree(sim.handle_scene, sim.object_shape_type, 2)
    local jointList = sim.getObjectsInTree(sim.handle_scene, sim.object_joint_type, 2)
    for i=1,#jointList,1 do
        objectList[#objectList + 1] = jointList[i]
    end
    local currHandle = -1
    for i=1,#objectList,1 do
        currHandle = objectList[i]
        childList = sim.getObjectsInTree(currHandle, sim.handle_all, 1)
        if #childList == 0 then
            sim.removeObject(currHandle)
        else
            recursiveRemove(currHandle)
        end
```

```
    end
    modelLoaded = false
    ROSSetup = false
    ROSCom = false
end


-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-- Funcion encargada de obtener las propiedades del modelo del robot
-------------------------------------------------------------------------------------------------------------
function getDimensions(objectHandle)
    --      xSize: Diametro del cilindro
    --      ySize: Diametro del cilindro (duplicado)
    --      zSize: Longitud del cilindro
    local localxMinMax={0,0}
    local localyMinMax={0,0}
    local localzMinMax={0,0}
    local result = nil
    result,localxMinMax[1]=sim.getObjectFloatParameter(objectHandle,15)
    result,localyMinMax[1]=sim.getObjectFloatParameter(objectHandle,16)
    result,localzMinMax[1]=sim.getObjectFloatParameter(objectHandle,17)
    result,localxMinMax[2]=sim.getObjectFloatParameter(objectHandle,18)
    result,localyMinMax[2]=sim.getObjectFloatParameter(objectHandle,19)
    result,localzMinMax[2]=sim.getObjectFloatParameter(objectHandle,20)
    local xSize=localxMinMax[2]-localxMinMax[1]
    local ySize=localyMinMax[2]-localyMinMax[1]
    local zSize=localzMinMax[2]-localzMinMax[1]
    return {xSize/2, zSize}
end
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-- Funcion responsable de la comunicación ROS
-------------------------------------------------------------------------------------------------------------
```

```
function startROS(u, id)
    if not modelLoaded then
        ROSCom = false
        out = string.format("Modelo no cargado! No se puede iniciar la comunicacion ROS!”)
simUI.setLabelText(ui, 1003, out)
    else
        ROSCom = true
        publishModelParam()
    end

end

function stopROS(u, id)
    ROSCom = false
end

-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-- setUpROS(): Configura los tópicos a manejarse en el programa
-------------------------------------------------------------------------------------------------------------
function setUpROS()
    robotHandle = sim.getObjectHandle("Delta_base")
    robotName = "Delta_base"

    --Verifique si la interfaz ROS esta disponible
    moduleName = 0
    index = 0
    rosInterfacePresent = false
    while moduleName do
        moduleName = sim.getModuleName(index)
        if(moduleName == 'RosInterface') then
            rosInterfacePresent = true
        end
```

```
        index = index+1
    end
    if rosInterfacePresent then
        revJointList                                                                    =
{sim.getObjectHandle("R1_B"),sim.getObjectHandle("R2_B"),sim.getObjectHandle("R3_
B")}
        revJointName = {}
        for i=1,#revJointList,1        do
            revJointName[i] = sim.getObjectName(revJointList[i]);
        end
    end

    jointpub = simROS.advertise('/Delta_base/rev_joint','sensor_msgs/JointState')
    eePospub = simROS.advertise('/Delta_base/ee_pos','geometry_msgs/Pose')
    timepub = simROS.advertise('/simulationTime','std_msgs/Float32')
    modepub = simROS.advertise('/eeMode','std_msgs/Bool')
    jointVelSub    =    simROS.subscribe('/Delta_base/joint_vel','sensor_msgs/JointState',
'jointVelSub_callback')

    baseParamPub = simROS.advertise('/Delta_base/base_radius','std_msgs/Float64')
    eeParamPub = simROS.advertise('/Delta_base/ee_radius','std_msgs/Float64')
    lowLinkParamPub = simROS.advertise('/Delta_base/lower_link','std_msgs/Float64')
    upLinkParamPub = simROS.advertise('/Delta_base/upper_link','std_msgs/Float64')

end

function publishModelParam()
    local base_values = getDimensions(base)
    local ee_values = getDimensions(sim.getObjectHandle("Delta_ee"))
    local lowlink_values = getDimensions(sim.getObjectHandle("Link1_1"))
    local uplink_values = getDimensions(sim.getObjectHandle("Link1_2"))

    simROS.publish(baseParamPub, {data = base_values[1]})
    simROS.publish(eeParamPub, {data = ee_values[1]})
```

```
    simROS.publish(lowLinkParamPub, {data = lowlink_values[2]})
    simROS.publish(upLinkParamPub, {data = uplink_values[2]})
end
------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------
-- Funcion responsable para configurar los mensajes ROS
------------------------------------------------------------------------------------------------------------------
function getHeader()
    return {header = {stamp=sim.getSystemTime()}}
end

function jointVelSub_callback(jointData)
    local name = jointData.name
    local velocity = jointData.velocity
    for i=1,#name,1
    do
        local handle = sim.getObjectHandle(name[i])
        sim.setJointTargetVelocity(handle, velocity[i])
    end
end

function jointsub_callback(jointData)
    local name = jointData.name
    local angle = jointData.position
    for i=1,#name,1
    do
        local handle = sim.getObjectHandle(name[i])
        sim.setJointTargetPosition(handle, angle[i])
    end
end

function getJointPositions()
    local posList = {}
    for i=1,#revJointList,1
```

```
    do
        posList[i] = sim.getJointPosition(revJointList[i]);
    end
    return posList
end

function getEEPos()
    local eeHandle = sim.getObjectHandle("Delta_ee")
    return sim.getObjectPosition(eeHandle, -1)
end

function changeMode()
    mode = not mode
end

-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
-- Función responsable para cargar el modelo del robot
-------------------------------------------------------------------------------------------------------------
function loadDelta(u, id)
    local filename = simUI.getEditValue(ui,2005)
    local f=io.open("deltaModels/"..filename)
    if f == nil then
        out = string.format("No existe archivo! Por favor coloque el nombre correcto.")
        simUI.setLabelText(ui, 1003, out)
    else
        io.close(f);
        local model = sim.loadModel("deltaModels/"..filename)
        out = string.format("Archivo existente!")
        simUI.setLabelText(ui, 1003, out)
        base = model
        -- sim.auxiliaryConsolePrint(consoleHandle, tostring(base)..'\n')
        -- sim.auxiliaryConsolePrint(consoleHandle, sim.getObjectName(base)..'\n')
```

```
        modelLoaded = true
    end
end

-----------------------------------------------------------------------------------------------------------
function closeEventHandler(h)
  sim.addStatusbarMessage('Window '..h..' is closing...')
  simUI.hide(h)
end

if (sim_call_type==sim.syscb_init) then

  xml = [[<ui closeable="false" on-close="closeEventHandler" resizable="true">
    <tabs>
      <tab title="Create Delta Robot">

        <group layout="vbox">
          <group layout="hbox">
            <button text="Limpiar escena" on-click="clearScene" id="2000" />
            <button text="Iniciar ROS COM" on-click="startROS" id="2001" />
            <button text="Detener ROS COM" on-click="stopROS" id="2002" />
            <button text="Cambiar Modo EE" on-click="changeMode" id="2003" />
            <button text="Cargar Modelo" on-click="loadDelta" id="2004" />
            <edit value="Nombre del modelo desde la carpeta raiz..."   id="2005" />
          </group>

          <label  text="<big>  Messages:</big>"  id="1002"  wordwrap="false"  style="font-
weight: bold;"/>
          <group layout="vbox">
           <label value="" id="1003" wordwrap="true" />
          </group>
          </group>
          <stretch />
    </tab>
```

```
    </tabs>
  </ui>]]
  -- consoleHandle = sim.auxiliaryConsoleOpen('Debug CVS', 200, 1)
  -- sim.auxiliaryConsolePrint(consoleHandle,"test\n")
  ui=simUI.create(xml)
  ROSCom = false
  ROSSetup = false
  modelLoaded = false
  mode = false

  local savedMode=sim.getInt32Parameter(sim.intparam_error_report_mode)
  sim.setInt32Parameter(sim.intparam_error_report_mode,0)

  local modeltest = sim.getObjectHandle('Delta_base')

  sim.setInt32Parameter(sim.intparam_error_report_mode,savedMode)

  if (modeltest ~= -1) then
      base = sim.getObjectHandle('Delta_base')
      modelLoaded = true
  else
      base = nil
      sim.auxiliaryConsolePrint(consoleHandle,"Modelo no encontrado!\n")
  end
end

if (sim_call_type==sim.syscb_actuation) then
   if modelLoaded and not ROSSetup then
      setUpROS()
      ROSSetup = true
   end
   if rosInterfacePresent and ROSCom then
      local jointInfo = getHeader()
      jointInfo["name"] = revJointName
```

```
        jointInfo["position"] = getJointPositions() -- Joint angles radianes

        local eeVal = getEEPos()
        local eePos = {}
        local eePoint = {}
        eePoint["x"] = eeVal[1]
        eePoint["y"] = eeVal[2]
        eePoint["z"] = eeVal[3]
        eePos["position"] = eePoint
        simROS.publish(timepub, {data = sim.getSimulationTime()})
        simROS.publish(jointpub, jointInfo)
        simROS.publish(modepub, {data = mode})
        simROS.publish(eePospub, eePos)
    end
end

if (sim_call_type==sim.syscb_sensing) then

    -- Put your main SENSING code here

end

if (sim_call_type==sim.syscb_cleanup) then

    -- Put some restoration code here

end
```

deltaParams.py

```
import rospy
from std_msgs.msg import Float64
from sensor_msgs.msg import JointState

class modelParams(object):
    def __init__(self):
        self.modelParams = dict.fromkeys(('rBase', 'rEE', 'lowLinkLength', 'upLinkLength'))
        self._baseParamSub = rospy.Subscriber("/Delta_base/base_radius", Float64, self.setBase)
        self._eeParamSub = rospy.Subscriber("/Delta_base/ee_radius", Float64, self.setEE)
        self._lowLinkParamSub = rospy.Subscriber("/Delta_base/lower_link", Float64, self.setLowerLink)
        self._upLinkParamSub = rospy.Subscriber("/Delta_base/upper_link", Float64, self.setUpperLink)
        self.num_param = 4
        self.updateCount = 0

    def loadIK(self):
        if self.updateCount == self.num_param:
            rospy.set_param('model_param',self.modelParams)
            rospy.loginfo("%d parametros cargados!", self.updateCount)
            self.updateCount = 0
            rospy.loginfo(getModelParams())
        else:
            rospy.loginfo("Parametros incompletos!. %d parametros cargados!", self.updateCount)

    def setBase(self,value):
        self.modelParams['rBase'] = value.data
        self.updateCount = self.updateCount + 1
```

```
        self.loadIK()

    def setEE(self,value):
        self.modelParams['rEE'] = value.data
        self.updateCount = self.updateCount + 1
        self.loadIK()

    def setLowerLink(self,value):
        self.modelParams['lowLinkLength'] = value.data
        self.updateCount = self.updateCount + 1
        self.loadIK()
    def setUpperLink(self,value):
        self.modelParams['upLinkLength'] = value.data
        self.updateCount = self.updateCount + 1
        self.loadIK()

def getModelParams():
    if rospy.has_param('model_param'):
        return rospy.get_param('model_param')
    else:
        rospy.logerr("Los parametros estan incompletos!")

def main():
    rospy.init_node("modelParamListener")
    listener = modelParams()
    rospy.spin()

if __name__=="__main__":
    main()
```

traj_control.py

```
#!/usr/bin/env python
import rospy
import numpy as np
import os
from std_msgs.msg import Bool
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Pose
from timeit import default_timer

import deltaModule as delta
import deltaParams as model

# Inicializa tópicos publicadores y subscriptores para el nodo delta_control.
# Suscribe el estado de las articulaciones del robot delta.
# Publica las velocidades de la articulación calculadas por el controlador.

class deltaControl(object):

   def __init__(self):
      self._e1PosPub = rospy.Publisher("/fi1", Float64, queue_size=1) # Publica el error de posicionamiento angular articulacion_1
      self._e2PosPub = rospy.Publisher("/fi2", Float64, queue_size=1) # Publica el error de posicionamiento angular articulacion_2
      self._e3PosPub = rospy.Publisher("/fi3", Float64, queue_size=1) # Publica el error de posicionamiento angular articulacion_3
     self._j1VelSub = rospy.Subscriber("/fo1", Float64, queue_size=1, self.loadRTTControl_1) # Recibe la velocidad calculada en Orocos de la articulación_1
      self._j2VelSub = rospy.Subscriber("/fo2", Float64, queue_size=1, self.loadRTTControl_2) # Recibe la velocidad calculada en Orocos de la articulación_2
```

```
        self._j3VelSub = rospy.Subscriber("/fo3", Float64, queue_size=1, self.
loadRTTControl_3) # Recibe la velocidad calculada en Orocos de la articulación_3
        self._jVelPub = rospy.Publisher("/Delta_base/joint_vel", JointState, queue_size=1)
# Publica la velocidad de la articulación (actuadores)
        self._jointSub = rospy.Subscriber("/Delta_base/rev_joint", JointState,
self.loadJointsPos) # Recibe la posicion angular de la articulacion
        self._modeSub = rospy.Subscriber("/eeMode", Bool, self.setMode)  # Recibe el
modo de operación

        #Vectores para almacenar los datos de las articulación recibidos
        self.jointNames = [] # Almacena el nombre de las articulaciones revoluta
        self.jointAngles = [] # Almacena la posición angular de las articulaciones
        self.jointControl = []# Almacena la velocidad angular de las articulaciones

        #Vector para almacenar los parámetros del robot delta
        self.modelParams = model.getModelParams()

        #Parametros de la trayectoria
        self.trajectory = 0
        self.counter = 0
        self.data_counter = 0
        self.traj_Loaded = False
        self.mode = False

        #Apertura del archivo para grabar datos
        self.file = open("data_rb.txt", "w")
        self.tiempo = 0

        #Tasa de publicacion del nodo en hercios
        self.pubRate = rospy.Rate(100)

    def loadJointsPos(self, joint_data):
        self.jointNames = joint_data.name
        self.jointAngles = joint_data.position
```

```
        self.control()

    def loadRTTControl_1(self, control_data):
        self.jointControl[0] = control_data.data

    def loadRTTControl_2(self, control_data):
        self.jointControl[1] = control_data.data

    def loadRTTControl_3(self, control_data):
        self.jointControl[2] = control_data.data

    def setMode(self, mode_data):
        self.mode = mode_data.data

    def control(self):
        if(self.traj_Loaded and not self.mode):
            #Agrega el tiempo al archivo para la toma de datos
            print("El tiempo actual es", self.tiempo)
            inicio = default_timer()#Variable de inicio para calcular el tiempo de ejecucion
            desiredX = self.trajectory[0][self.counter]
            desiredY = self.trajectory[1][self.counter]
            desiredZ = self.trajectory[2][self.counter]
            #Transforma el punto de la trayectoria deseada en posición angular utilizando la
cinemática inversa
            trajPos = np.transpose(delta.ik_delta(desiredX, desiredY , desiredZ,
self.modelParams))
            #Tranpuesta del vector posición angular actual de las articulaciones revolutas.
            curAng = np.transpose(self.jointAngles)

            #Cálculo del error entre la posición angular deseada y la posición angular actual.
            errorAng = trajPos - curAng
            errorq1 = Float64()
            errorq2 = Float64()
            errorq3 = Float64()
```

```
        errorq1.data = errorAng[0]
        errorq2.data = errorAng[1]
        errorq3.data = errorAng[2]

        #Creacion de un mensaje tipo JointState que se utilizara para publicar la velocidad del controlador en Orocos
        control_vel[0]= self.jointControl[0]
        control_vel[1]= self.jointControl[1]
        control_vel[2]= self.jointControl[2]
        controlInfo = JointState()
        controlInfo.name = self.jointNames
        controlInfo.velocity = [control_vel[0], control_vel[1], control_vel[2]]
        self._jVelPub.publish(controlInfo)

        #Actualiza la posicion de la trayectoria
        self.counter += 1
        if self.counter == len(self.trajectory[0]):
            self.counter = 0
        #print (self.data_counter)
        if self.data_counter < 500:
            self.file.write(str(self.tiempo)+",")
            self.file.write(str(trajPos[0]) + "," + str(trajPos[1]) + "," + str(trajPos[2])+ ",")
            self.file.write(str(curAng[0]) + "," + str(curAng[1]) + "," + str(curAng[2]) + "\n")
            self.data_counter += 1
        #Publica el mensaje. Espera cumplir el periodo de la simulación del sistema de control 100Hz.
        self.pubRate.sleep()

        #Calcula el tiempo de ejecucion
        fin = default_timer()#Variable de finalizacion para calcular el tiempo de ejecucion
        self.tiempo = self.tiempo + (fin - inicio)#Calculo del tiempo de ejecucion

    # Funcion que construye la trayectoria que el efector final del robot seguirá.
    def setUpTraj(self):
```

```
        #Actualiza la trayectoria radio 0.25
        radius = .25
        traj = [[], []]
        rads = np.linspace(0, 2 * np.pi, 250)
        for i in range(len(rads)):
            traj[0].append(np.cos(rads[i]) * radius)
            traj[1].append(np.sin(rads[i]) * radius)
        traj[0].extend(traj[0][::-1])
        traj[1].extend(traj[1][::-1])
        self.trajectory = traj
        self.traj_Loaded = True

def main():
    rospy.init_node('delta_control')
    deltaObject = deltaControl()
    deltaObject.setUpTraj()
    rospy.spin()

if __name__=="__main__":
    main()
```

deltaModule.py

```
import numpy as np
from math import cos, sin, radians, atan, pi

def distance(a, b):
    return np.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2 + (a[2] - b[2]) ** 2)

def ik_delta(x, y, z, modelParams):
    p = np.array([x, y, z])
    blist = np.array([[modelParams['rBase'], 0, 0],
                      [modelParams['rBase'] * cos(radians(120)), modelParams['rBase'] *
sin(radians(120)), 0],
```

```
                [modelParams['rBase'] * cos(radians(240)), modelParams['rBase'] *
sin(radians(240)), 0]])
    plist = np.array([[modelParams['rEE'], 0, 0],
                [modelParams['rEE'] * cos(radians(120)), modelParams['rEE'] *
sin(radians(120)), 0],
                [modelParams['rEE'] * cos(radians(240)), modelParams['rEE'] *
sin(radians(240)), 0]])

    pmblist = plist - blist
    bmplist = blist - plist

    G = np.zeros((1, 3))
    E = np.zeros((1, 3))
    F = np.zeros((1, 3))
    tp = np.zeros((1, 3))
    tm = np.zeros((1, 3))
    thetap = np.zeros((1, 3))
    thetam = np.zeros((1, 3))
    theta = [None, None, None]

    try:
        for i in range(3):
            G[0][i] = modelParams['upLinkLength'] ** 2 - modelParams['lowLinkLength'] ** 2 -
distance(p, bmplist[i]) ** 2
            E[0][i] = 2 * p[2] * modelParams['lowLinkLength'] + 2 * pmblist[i][2] *
modelParams['lowLinkLength']
            F[0][i] = 2 * modelParams['lowLinkLength'] * (
                (p[0] + pmblist[i][0]) * cos(radians((i + 1) * 120 - 120)) + (p[1] + pmblist[i][1]) *
sin(
                    radians((i + 1) * 120 - 120)))
            tp[0][i] = (-F[0][i] + np.sqrt(E[0][i] ** 2 + F[0][i] ** 2 - G[0][i] ** 2)) / (G[0][i] - E[0][i])
            tm[0][i] = (-F[0][i] - np.sqrt(E[0][i] ** 2 + F[0][i] ** 2 - G[0][i] ** 2)) / (G[0][i] - E[0][i])
            thetap[0][i] = 2 * atan(tp[0][i])
            thetam[0][i] = 2 * atan(tm[0][i])
```

```
            if -pi / 4 <= thetap[0][i] <= pi / 2:
                print np.isreal(thetap[0][i])
                if np.isreal(thetap[0][i]):
                    theta[i] = thetap[0][i] * -1

            if -pi / 4 <= thetam[0][i] <= pi / 2:
                if np.isreal(thetam[0][i]):
                    theta[i] = thetam[0][i] * -1
    except RuntimeError:
        return np.array(theta)
    return np.array(theta)

if __name__=="__main__":
    print "ERROR"
```

Orocos

Inicio.ops

```
import("rtt_ros")
ros.import("robot_control")

## Carga el componente principal
loadComponent("controlador","Main")

## Configura la actividad periódica
setActivity("controlador",0.0,HighestPriority,ORO_SCHED_RT)

## Crea las conexiones
stream("controlador.fi_1", ros.topic("fi1"))
stream("controlador.fi_2", ros.topic("fi2"))
stream("controlador.fi_3", ros.topic("fi3"))
stream("controlador.fo_1", ros.topic("fo1"))
stream("controlador.fo_2", ros.topic("fo2"))
stream("controlador.fo_3", ros.topic("fo3"))
```

```
stream("controlador.string_out", ros.topic("string_out"))
stream("controlador.string_in", ros.topic("string_in"))

## Conecta el servicio ROS
loadService("controlador","rosservice")
controlador.rosservice.connect( "increment", "/increment", "std_srvs/Empty")
controlador.rosservice.connect( "updated", "/updated", "std_srvs/Empty")

## Configura e inicia el componente
controlador.configure()
controlador.start()
```

Control.cpp

```
#include <rtt/TaskContext.hpp>
#include <rtt/Port.hpp>
#include <std_msgs/Float64.h>
#include <std_msgs/String.h>
#include <rtt/Component.hpp>
#include <std_srvs/Empty.h>
#include <ros/ros.h>
#include <math.h>

using namespace RTT;

class Main : public RTT::TaskContext{
private:
  InputPort<std_msgs::Float64> input_1;
  InputPort<std_msgs::Float64> input_2;
  InputPort<std_msgs::Float64> input_3;
  OutputPort<std_msgs::Float64> output_1;
  OutputPort<std_msgs::Float64> output_2;
  OutputPort<std_msgs::Float64> output_3;
```

```
  InputPort<std_msgs::String> sinport;
  OutputPort<std_msgs::String> soutport;

  std::string prop_answer;
  double prop_counter_step;
  double prop_service_call_counter;
  double counter;
  double edata[3];
  double eInt[3];
  double eprev[3];
  double Derror[3];
  double dt;

  double Kp;
  double Ki;
  double Kd;

  OperationCaller<bool(std_srvs::Empty::Request&,        std_srvs::Empty::Response&)>
updated;

public:
  Main(const std::string& name):
    TaskContext(name),

input_1("fi_1"),input_2("fi_2"),input_3("fi_3"),output_1("fo_1"),output_2("fo_2"),output_3(
"fo_3"),
    sinport("string_in"),soutport("string_out","Hello Robot"),
    prop_answer("Hello Robot"),prop_counter_step(0.01),prop_service_call_counter(0.0),
    updated("updated")
  {
    this->addEventPort(input_1).doc("Receiving  a  message  here  will  wake  up  this
component.");
    this->addEventPort(input_2).doc("Receiving  a  message  here  will  wake  up  this
component.");
```

```
    this->addEventPort(input_3).doc("Receiving a message here will wake up this component.");
    this->addPort(output_1).doc("Sends out 'answer'.");
    this->addPort(output_2).doc("Sends out 'answer'.");
    this->addPort(output_3).doc("Sends out 'answer'.");
    this->addEventPort(sinport).doc("Receiving a message here will wake up this component.");
    this->addPort(soutport).doc("Sends out a counter value based on 'counter_step'.");

    this->addProperty("answer",prop_answer).doc("The text being sent out on 'string_out'.");
    this->addProperty("counter_step",prop_counter_step).doc("The increment for each new sample on 'float_out'");
    this->addProperty("service_call_counter",prop_service_call_counter).doc("The number of times the incrememt operation has been called.");

    this->provides()->addOperation("increment",&Main::increment,this,RTT::OwnThread);
    this->requires()->addOperationCaller(updated);

    counter=0.0;

    eInt[0]=0;
    eInt[1]=0;
    eInt[2]=0;
    eprev[0]=0;
    eprev[1]=0;
    eprev[2]=0;
    Derror[0]=0;
    Derror[1]=0;
    Derror[2]=0;
    Kp=400;
    Ki=10;
    Kd=1;
    dt=0.01;
```

```
    edata[0]=0;
    edata[1]=0;
    edata[2]=0;
  }
  ~Main(){}
private:
  void updateHook(){
    std_msgs::Float64 fdata_1;
    std_msgs::Float64 fdata_2;
    std_msgs::Float64 fdata_3;
    std_msgs::Float64 udata_1;
    std_msgs::Float64 udata_2;
    std_msgs::Float64 udata_3;
    std_msgs::Float64 resp;
    std_msgs::String sdata;

    if(NewData==input_1.read(fdata_1)){
      log(Info)<<"error q1: "<<fdata_1<<endlog();
      edata[0]=fdata_1.data;
      eInt[0]+=edata[0];
      Derror[0]=(edata[0]-eprev[0])/dt;
      eprev[0]=edata[0];
      udata_1.data=(Kp*edata[0]+Ki*eInt[0]+Kd*Derror[0])*(M_PI/180);
      output_1.write(udata_1);
    }

    if(NewData==input_2.read(fdata_2)){
      log(Info)<<"error q2: "<<fdata_2<<endlog();
      edata[1]=fdata_2.data;
      eInt[1]+=edata[1];
      Derror[1]=(edata[1]-eprev[1])/dt;
      eprev[1]=edata[1];
      udata_2.data=(Kp*edata[1]+Ki*eInt[1]+Kd*Derror[1])*(M_PI/180);
```

```
    output_2.write(udata_2);
   }

   if(NewData==input_3.read(fdata_3)){
    log(Info)<<"error q3: "<<fdata_3<<endlog();
    edata[2]=fdata_3.data;
    eInt[2]+=edata[2];
    Derror[2]=(edata[2]-eprev[2])/dt;
    eprev[2]=edata[2];
    udata_3.data=(Kp*edata[2]+Ki*eInt[2]+Kd*Derror[2])*(M_PI/180);
    output_3.write(udata_3);
   }

   if(NewData==sinport.read(sdata))
    log(Info)<<"String in: "<<sdata<<endlog();
    soutport.write(sdata);

   if(updated.ready()) {
    std_srvs::Empty::Request req;
    std_srvs::Empty::Response res;
    updated(req,res);
   }
  }

  bool increment(std_srvs::Empty::Request& req, std_srvs::Empty::Response& resp) {
   prop_service_call_counter++;
   return true;
  }
};
ORO_CREATE_COMPONENT(Main)
```