TELEOPERACIÓN DE ROBOTS HUMANOIDES POR IMITACIÓN

Autor: Sammy Pfeiffer
Coautor: Toni Gabás

Agradecimientos a:
AESS Estudiants como asociación que ha cedido un Kinect para este proyecto y a sus miembros por estar siempre dispuestos a ayudar.
Cecilio Angulo y Manuel Velasco por permitir el uso del robot Nao y en general apoyar este proyecto.
Wataru Yoshizaki por ser la inspiración a retomar este trabajo.
Toni Gabás por su inestimable colaboración.

Teleoperación de robots humanoides por imitación

Resumen
En este trabajo se trata de imitar el trabajo de Wataru Yoshizaki donde teleopera un robot humanoide a partir del control de fuerzas de cada uno de sus servos. En nuestro caso usaremos un robot Nao con un Kinect para capturar el movimiento del usuario.
Se exploran diferentes posibilidades a lo largo del desarrollo del proyecto para la realización de este.
La aplicación final en mente de este proyecto es conseguir hacer luchar a dos robots teleoperados.

Índice de contenido

Parte 1: Fundamentos / State of the art

Por Wataru Yoshizaki:

An Actuated Physical Puppet as an Input Device for Controlling a Digital Manikin (enlace):
En este artículo se explica el sistema para usar una marioneta robótica para controlar la pose de un personaje virtual. Se podría tomar como la mitad del enfoque necesario para este proyecto.

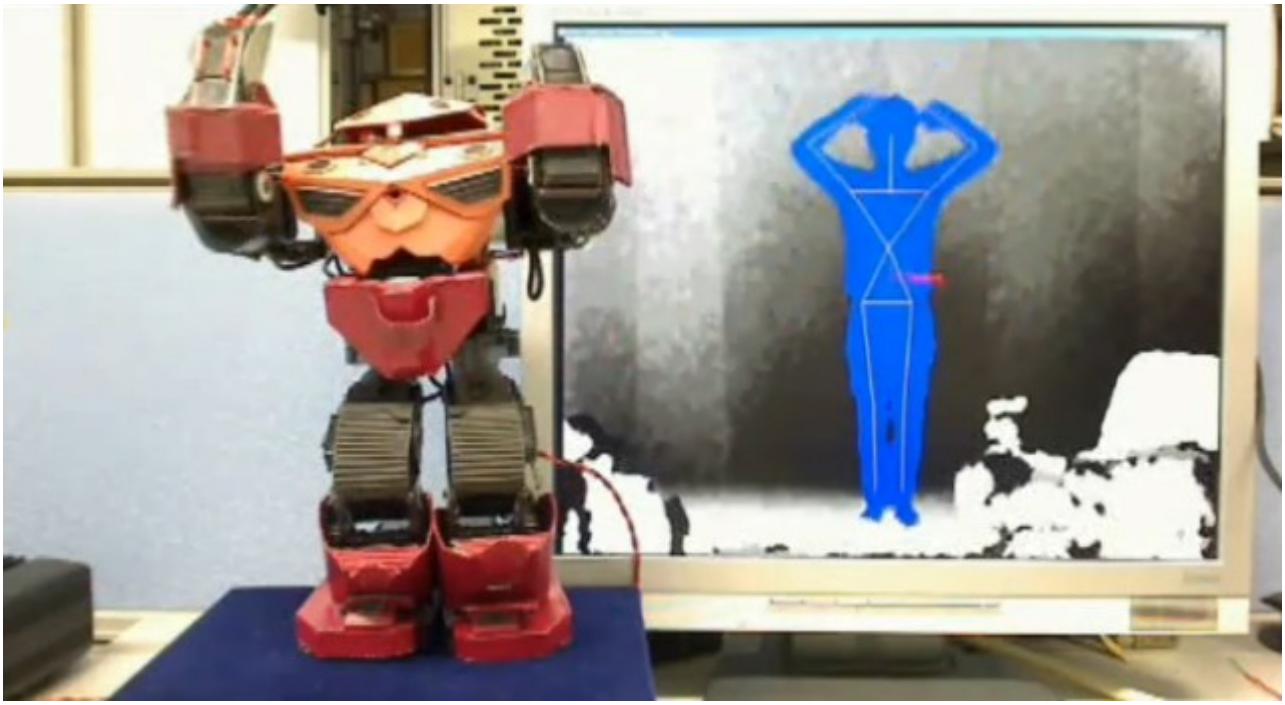

Illustration 1: V-sido con Kinect moviendo un robot.

A través de su página web del proyecto V-sido y gracias a su visita a la facultad de industriales para dar una charla hemos podido saber más de como realiza su trabajo. En su web se pueden encontrar videos de su software funcionando.
Concretamente usa el amperaje que pasa por cada uno de los servos del robot para tener una idea aproximada de la fuerza que está haciendo cada servo del robot. Con esta información junto con la posición de cada servo se puede obtener una imágen del estado dinámico del sistema.
A partir de estos datos se pueden ejecutar ordenes de posición en el robot tratando de equilibrarlo mientras se cumplen estas ordenes.

Por mi, Sammy Pfeiffer:

En el documento de mi proyecto final de carrera (enlace) podemos ver un planteamiento parcial de lo que se pretende hacer en este trabajo.

Illustration 2: Software del proyecto final de carrera moviendo el robot Bioloid.

Solo implica a los brazos. Más adelante se añadió la inclinación del pecho. Todo esto ignorando la dinámica del sistema, sin tratar de equilibrar el robot.
En este proyecto se aplica una orden de posición equivalente a la del usuario a cada una de las posiciones de las manos del robot en referencia a si mismo. Dicho de otro modo, cuando el usuario pone una mano enfrente de su pecho, se calcula a que distancia está del plano del pecho, se calcula el punto equivalente en el robot y se le da la orden de poner la mano del robot (a través de cinemática inversa) en esta posición.

Robot Nao:

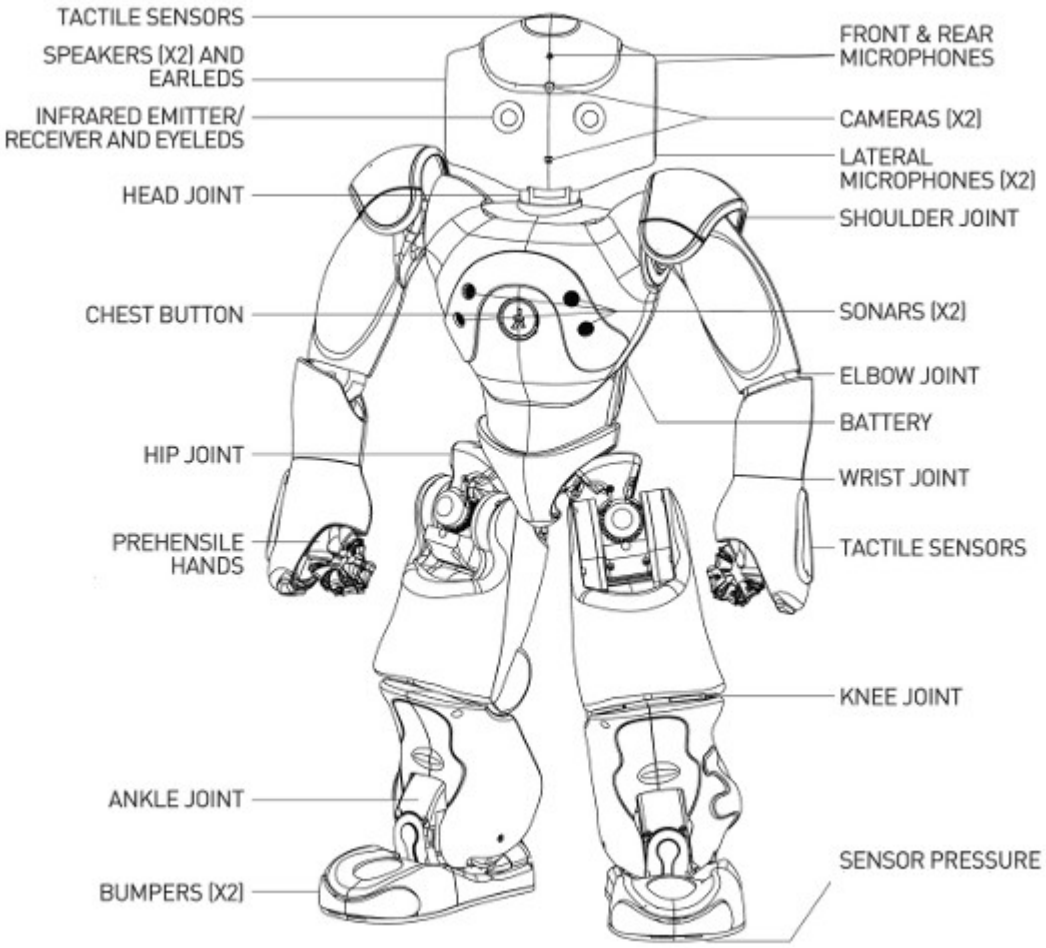

Illustration 3: Robot Nao de Aldebaran Robotics.

Nao es un robot humanoide de la empresa Aldebaran Robotics que actualmente (versión 4.0) consta de:
-25 grados de libertad gracias a sus motores y actuadores.
-Red de sensores que incluye:
- -2 cámaras
- -4 microfonos
- -1 sonar
- -2 emisores/receptores de infrarrojos
- -1 placa inercial
- -9 sensores tactiles
- -8 sensores de presión

-Procesador Intel Atom a 1.6Ghz con un sistema operativo Linux
-Batería con autonomía de unas 1.5 horas.

Este robot se ofrece a universidades o desarrolladores independientes para

investigación.
En nuestro caso hemos usado la versión anterior de Nao (versión 3.3) la cual se diferencia principalmente en la capacidad del procesador. Este pasa a ser un AMD a 500Mhz.

Nao se programa a través de varios lenguajes (C++, Python, Java, Matlab...) usando NaoQi. NaoQi es el framework puente intermedio entre el hardware del robot y la programación de él.
Además consta de Choreographe para programar el robot visualmente (con programación de tipo "cajas").
También consta de un simulador asociado, Webots, de una compañía externa.

Kinect:

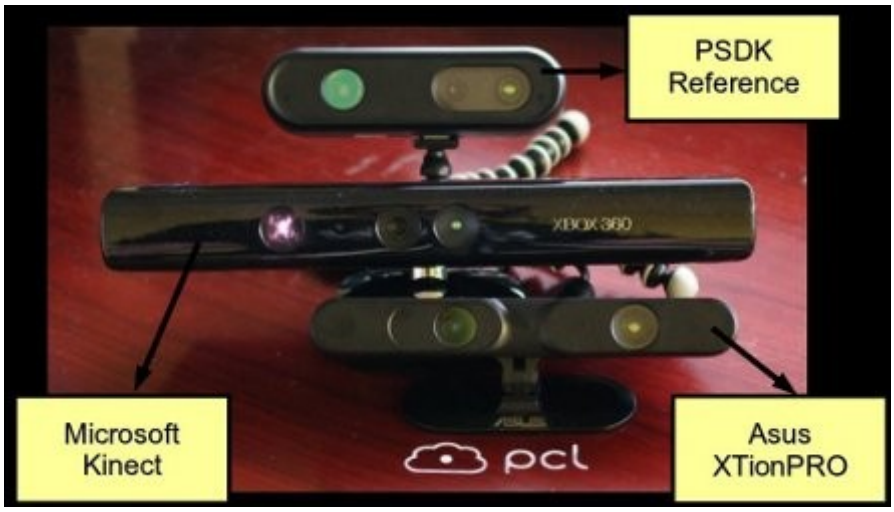

Illustration 4: Diferentes sensores tipo Kinect.

Parafraseando a la Wikipedia encontramos que Kinect se puede definir como: "un controlador de juego libre y entretenimiento desarrollado por Microsoft para la videoconsola Xbox 360. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz, y objetos e imágenes ".
Kinect se compone principalmente de:
-Una cámara tradicional (Resolución 640x480 RGB 30fps VGA).
-Un emisor de infrarrojos.
-Una cámara de infrarrojos.
-4 Micrófonos (16bit sampling rate: 16Hz).
-Un motor.

Gracias a él podemos obtener una imágen de profundidad de la escena y a partir de esta podemos conseguir un esqueleto del usuario que haya delante del sensor.

Illustration 5: Esqueleto capturado gracias a Kinect.

ROS:

Illustration 6: Logo de ROS.

ROS (Robotic Operating System) es un framework para desarrollo de software para robots. ROS provee los servicios estándar de un sistema operativo tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Esto está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar sensores, controlar, controlar estados, planificar, actuar y otros mensajes. La librería está orientada a usarse en un sistema UNIX.

Además consta de un simulador (gazebo) y un visor de estado multifuncional (Rviz) para monitorizar el estado de cualquier robot y casi cualquier variable publicada en el entorno de ROS.

Dentro de ROS existe un stack (paquete) para el robot Nao gracias al cualquier se puede obtener una interacción básica con el robot:
http://www.ros.org/wiki/nao_robot

Consta de unos nodos que interactúan con NaoQi.

Parte 2: Desarrollo

El desarrollo de este proyecto, al ser de carácter práctico y experimental, consta de diferentes pequeñas pruebas que sirven para llegar al objetivo final.

Pruebas kinect en ROS

Uno de los primeros elementos necesarios para el proyecto es la integración y familiarización de Kinect en ROS.

Afortunadamente hoy día esto es fácil:

Gracias al paquete openni_kinect podemos obtener acceso a los datos proporcionados por varias cámaras de tipo Kinect (Asus Xtion y Xtion Pro por ejemplo) de forma sencilla.

Y en el paquete openni_tracker encontraremos la herramienta de NITE que permite realizar skeleton tracking.

Es tan fácil como (con una instalación de ROS versión "fuerte" funcionando) ejecutar en un terminal:

rosrun openni_tracker openni_tracker

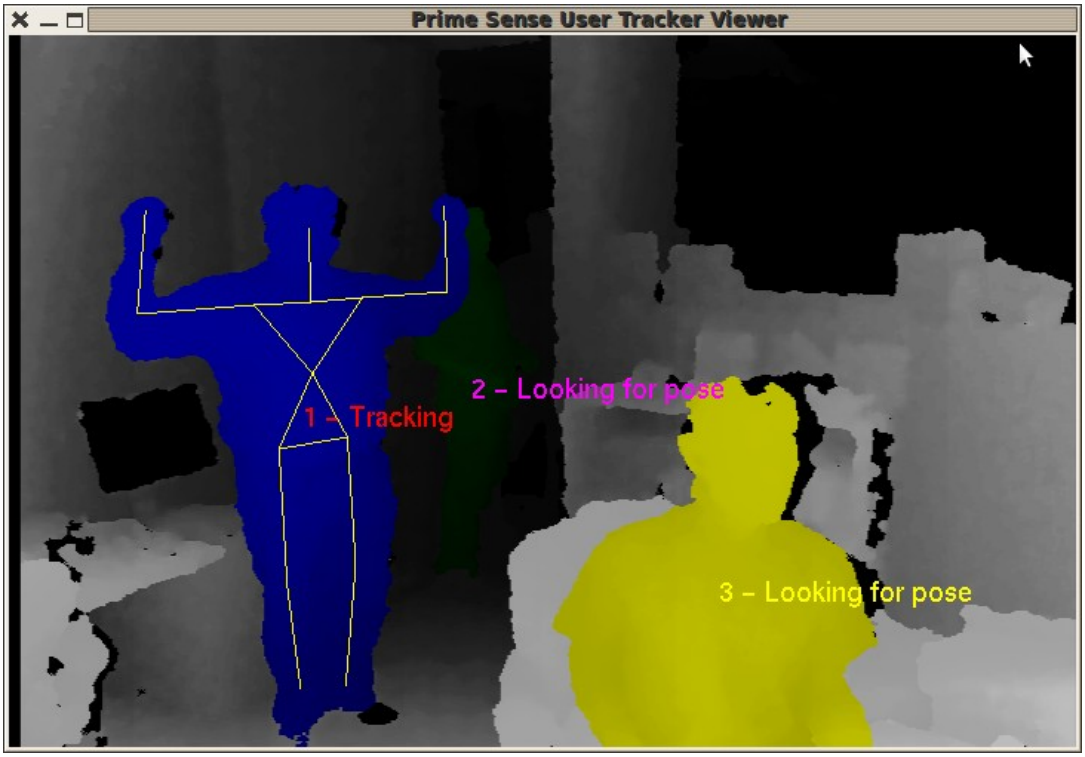

Illustration 7: openni_tracker de ROS funcionando.

Para ver los datos de los puntos del esqueleto del usuario en Rviz podemos usar el paquete skeleton_markers el cual aprovecha la posición publicada en

un nodo de ROS llamado TF, esto se puede hacer ejecutando en un terminal:
roscd skeleton_markers

rosrun rviz rviz -d markers.vcg

roslaunch skeleton_markers markers.launch

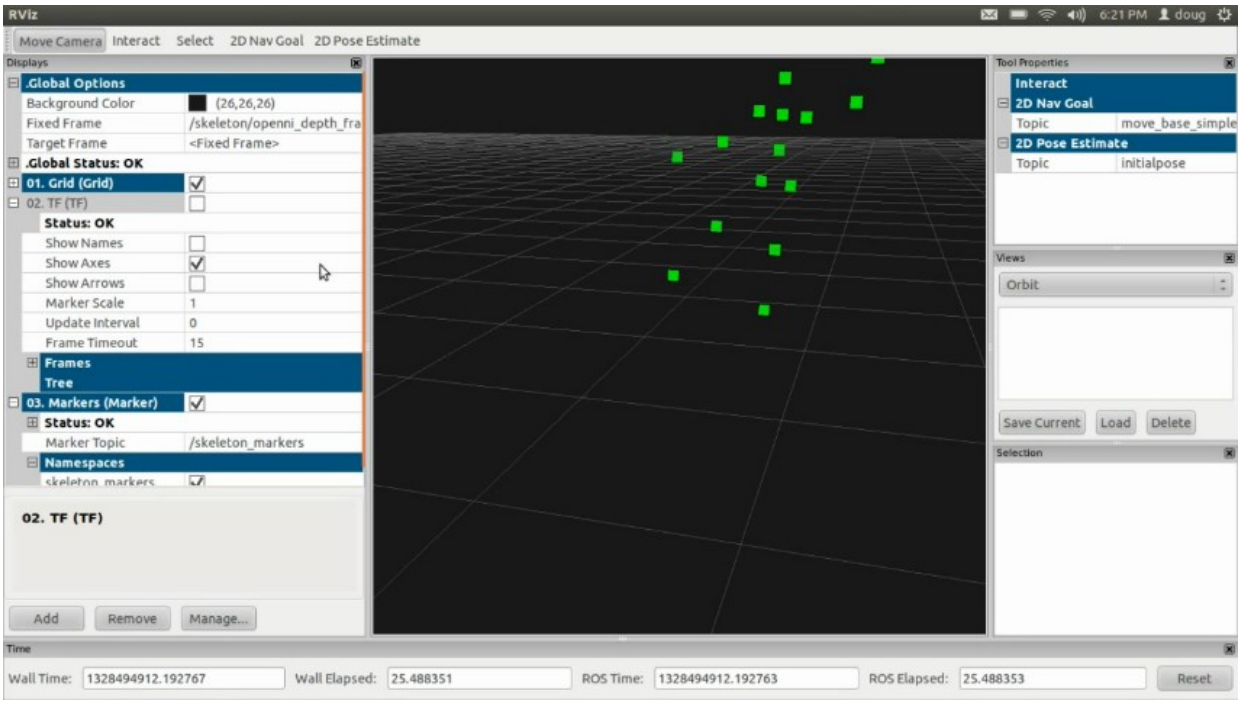

Illustration 8: skeleton_markers en Rviz.

El esqueleto consta de 15 puntos:

head	neck	torso
r_shoulder	r_elbow	r_hand
l_shoulder	l_elbow	l_hand
r_hip	r_knee	r_foot
l_hip	l_knee	l_foot

Pruebas con Nao

Ha hecho falta familiarizarse con el robot, su entorno de programación y ver sus limitaciones, con lo cual se han seguido los siguientes pasos.

Testeo de habilidades

Primeramente instalamos Chroeographe y lo abrimos.

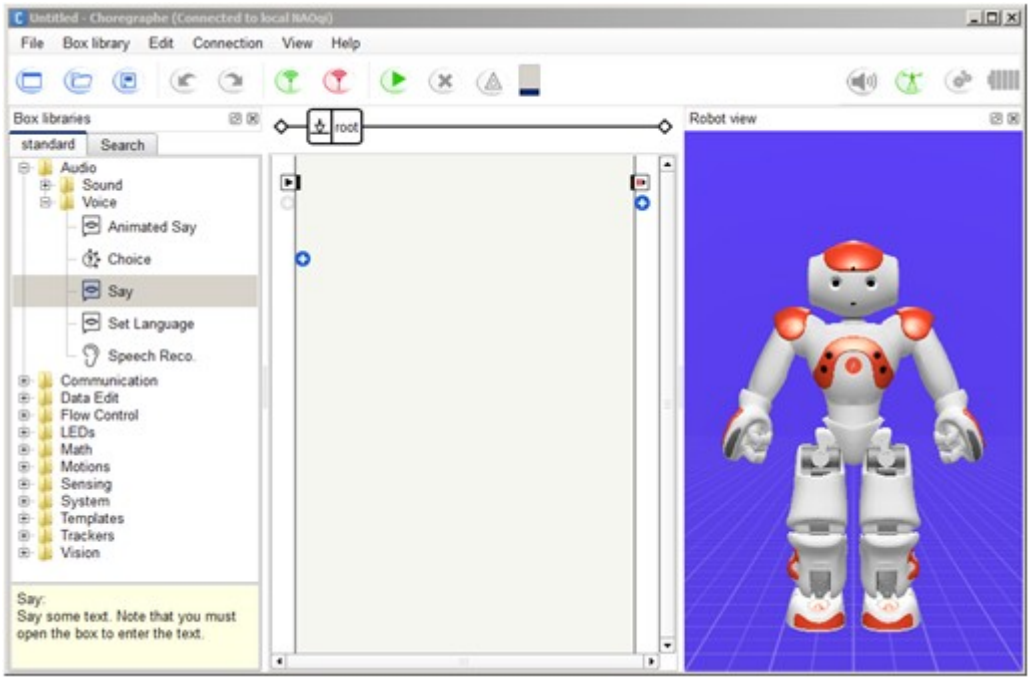

Illustration 9: Interfaz de Choreographe.

Y probamos los diferentes movimientos predefinidos que nos proporcionan.

Para nuestra sorpresa una de las demos que consiste en inclinar el robot hacia delante en equilibrio... dejaba caer el robot al suelo. No es nada grave la caída dado que el robot detecta esta caída y pone los brazos para cubrir su cabeza y sensores. Pero deja entrever que el autoequilibrado no funciona demasiado bien.

A continuación probamos el SDK en python para programar contra NaoQi, el framework de Nao.

Empezamos mandando ordenes de movimientos cartesianos a un brazo siguiente este tutorial: http://www.aldebaran-robotics.com/documentation/dev/python/examples/motion/cartesian.html#python-example-motion-cartesian

Y a continuación probamos las ordenes de movimiento con el controlador "whole body motion" siguiente este otro tutorial: http://www.aldebaran-robotics.com/documentation/dev/python/examples/motion/whole_body.html#

python-example-motion-wholebody

Observamos que la programación es fácil de comprender y consta de pocas ordenes en ambos casos para conseguir nuestro objetivo.

Descubrimiento de límites de la plataforma

Al haber diferentes maneras de mover la estructura del robot se han probado los diferentes sistemas y sacado sus pros y sus contras.

WbSetEffectorControl

Este método es el más interesante de mover el robot dado que promete el equilibrado del robot al ejecutar movimientos. Es muy sencillo de usar:

wbSetEffectorControl(effectorName, targetCoordinate)

Donde effectorName es el nombre de la parte del cuerpo a mover y targetCoordinate el punto del espacio donde moverlo.

Ventajas:

-Al ordenar un movimiento este se ejecuta manteniendo el equilibrio del robot.

Inconvenientes:

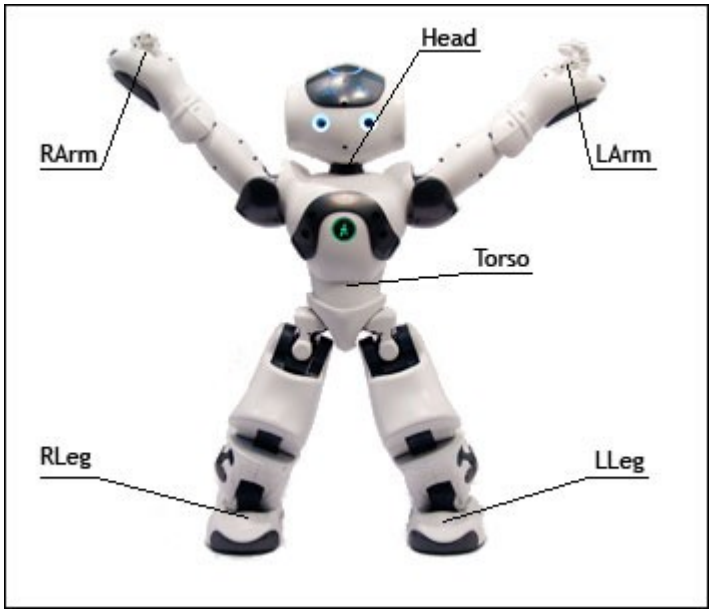

Illustration 10: Effectors de Nao.

-Solo hay 6 effectors como refleja la imagen. Con lo cual no podemos elegir donde poner los codos o las rodillas, por ejemplo.

-Solo podemos mandar 5 ordenes por segundo. Es un sistema muy lento.

setPosition(chainName, space, position, fractionMaxSpeed, axisMask)

Este método controla la posición de un effector (chainName en este caso) sin equilibrar el robot.

Ventajas:

-Más rápido, mayor número de ordenes por segundo (30 ordenes experimentalmente).

Inconvenientes:

-Sin equilibrado.

-Solo se puede elegir la posición de los 6 effectors.

-Falta de feedback al usar este método. Cuando la cinemática inversa falla (al darle una orden fuera del rango que puede cumplir o en una singularidad) se bloquea y tarda unos segundos en volver a procesar ordenes.

changePosition(effectorName, space, positionChange, fractionMaxSpeed, axisMask)

Este método desplaza un effector el cambio designado por positionChange.

Ventajas:

-Todavía más rápido que los métodos anteriores (40 ordenes por segundo experimentalmente).

Inconvenientes:

-Sin equilibrado.

-Solo se puede elegir la posición de los 6 effectors.

setAngles(names, angles, fractionMaxSpeed)

Este método mueve los servos (names) a las posiciones (angles) designadas a la velocidad (fractionMaxSpeed).

Ventajas:

-El más rápido de todos los métodos, en un ciclo de lectura de posición y escritura podemos bajar hasta a 10ms, 100 ordenes por segundo.

-Podemos situar cualquier parte del robot donde sea necesario. No estamos limitados por los effectors.

Inconvenientes:

-Sin equilibrado.

-Prescindimos de prácticamente todo lo implementado en el robot, hay que reimplementar la cinemática y dinámica del robot.

Pruebas de interfaz Nao en ROS

Hemos tenido que instalar el stack nao_robot de ROS. La instalación es sencilla y está descrita aquí: http://www.ros.org/wiki/nao/Installation

Para correr la simulación hará falta ejecutar los siguientes comandos en terminales separados:

roscore

~/NAO/naoqi-sdk-1.14-linux64$ ./naoqi --verbose --broker-ip 127.0.0.1

LD_LIBRARY_PATH=~/NAO/naoqi-sdk-1.14-linux64/lib: $LD_LIBRARY_PATH NAO_IP=127.0.0.1 roslaunch nao_driver nao_driver_sim.launch

roslaunch nao_description nao_state_publisher.launch

Con esto tendremos corriendo el servidor NaoQi que hace de robot Nao y los nodos que hacen de intermediarios entre NaoQi y ROS.

Ahora podremos abrir Rviz con:

rosrun rviz rviz

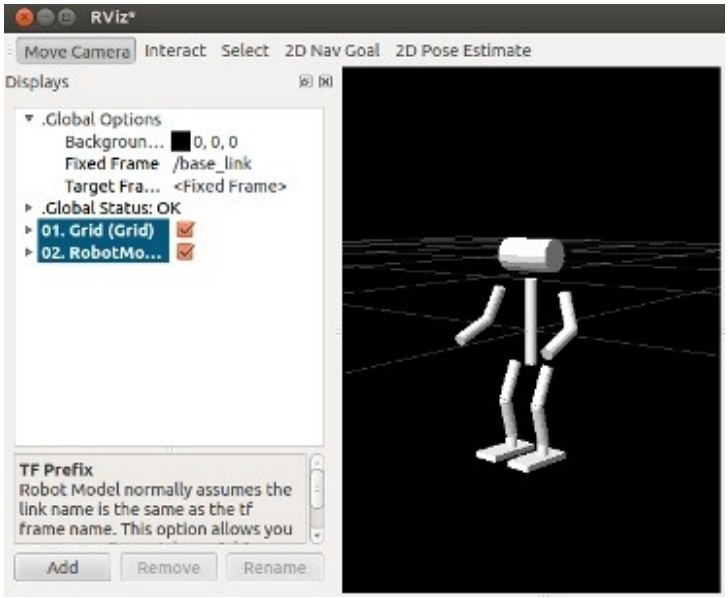

Illustration 11: Modelo de Nao en rviz.

Y ver el estado del Nao.

Podemos cambiar la posición del Nao abriendo Choreographe o usando el SDK y programando movimientos.

Mejora del modelo de simulación de Nao

Dado que el modelo del robot que podemos visualizar es bastante sencillo decidimos mejorarlo implementando texturas del robot.

Para esto se tuvo que conseguir cada una de estas texturas.

Se emprendieron varios caminos para conseguirlas.

En primer lugar a través del proyecto final de carrera de Jorge Bermejo Juárez donde podemos encontrar un repositorio donde tiene un modelo del robot subido. En este proyecto dice haber realizado estas piezas él mismo aunque sospechosamente he encontrado unas piezas exactamente iguales en un proyecto más antiguo.

Por otro lado se siguió el camino de la ingeniería inversa del simulador Webots for Nao. Este camino fue infructuoso dado a la complicación de extraer el modelo del robot ya que está creado en un lenguaje VRML 2.0 usando un tag llamado PROTO. Hoy día es muy difícil encontrar herramientas que trabajen con este formato. Se tuvo que remontar a herramientas para Windows Millemium que entendieran este formato, y tras aplicar muchos cambios manuales a los archivos, se consiguió cargar el modelo y exportarlo a formatos más amigables. Estos modelos resultaron tener una calidad bastante inferior a los conseguidos por el otro método.

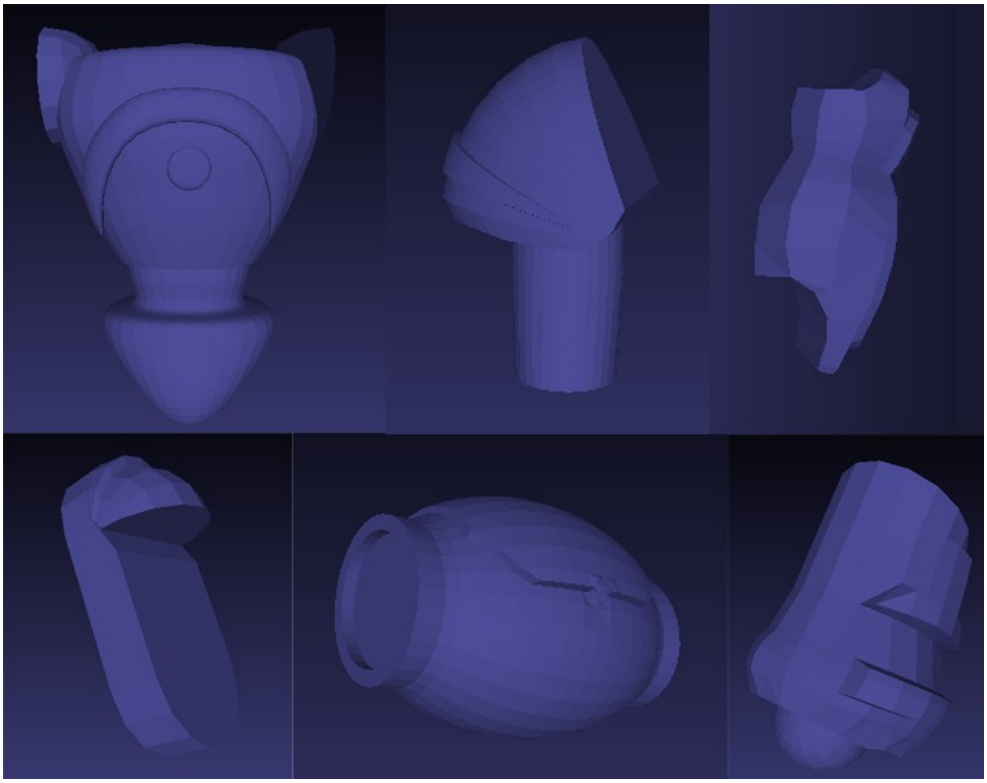

Illustration 12: Meshes de las partes del cuerpo del Nao.

Para conseguir usar estos modelos en la simulación se han tenido que modificar los archivos urdf del modelo del package de nao_common para que cargaran estas texturas.

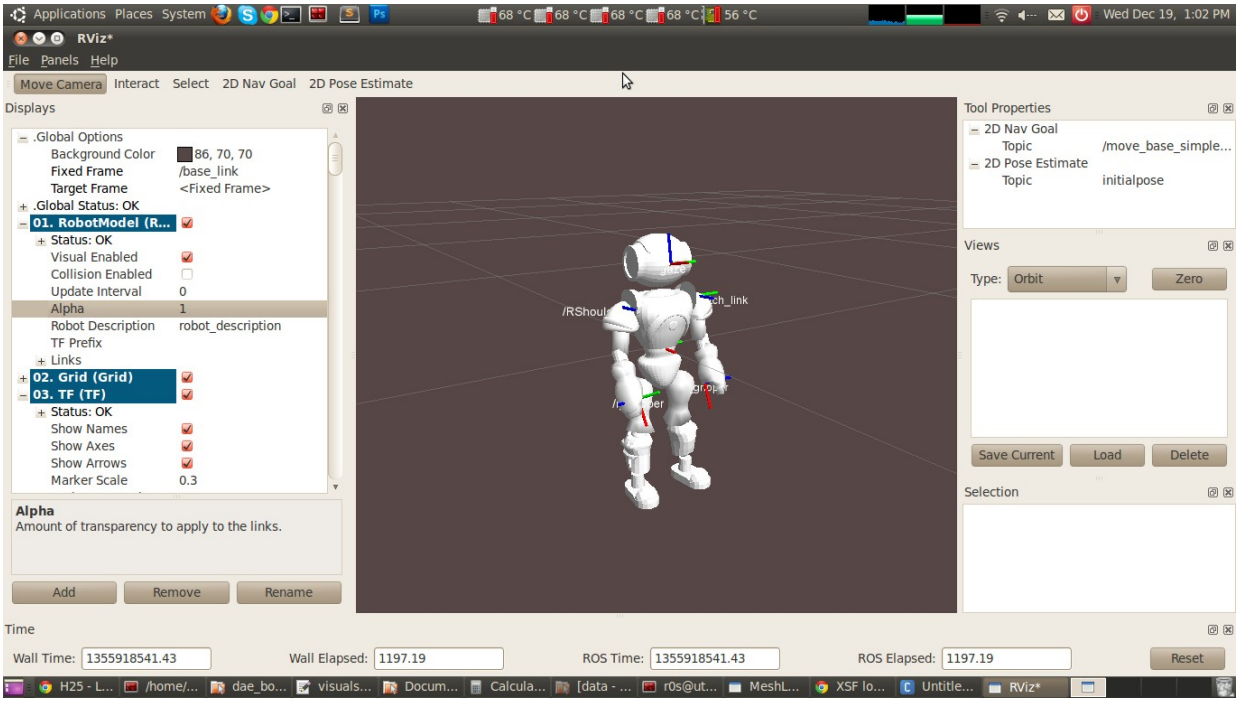

Illustration 13: Modelo de Nao corriendo en rviz.

Movimiento de un brazo con teclado y raton

Con el fin de probar los diferentes métodos de movimiento del robot se ha creado una aplicación simple de movimiento de un brazo del Nao.

Con las teclas direccionales del teclado se mueve la mano del robot en el plano X, Y. Con el ratón haciendo scroll up & down se controla el eje Z. Con los botones extra del ratón (movimiento de la rueda del ratón hacia los lados y botones laterales para el pulgar arriba y abajo) se controla la inclinación del robot sobre los ejes X e Y.

En el anexo correspondiente se puede encontrar el código.

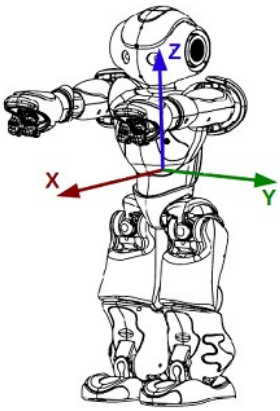

Illustration 14: Ejes de Nao.

Conjunción de datos de esqueleto kinect con robot

Para poder visualizar los datos del esqueleto y del robot al mismo tiempo en el simulador hay que conseguir enlazar la información de ambos, publicado en TF, en una transformada común.

Para ello basta con ejecutar:
rosrun tf static_transform_publisher 0.0 0.5 0.0 0.0 0.0 0.0 openni_depth_frame base_link 100
Lo cual publica una transformada estática entre la referencia de posición del Kinect y el robot.

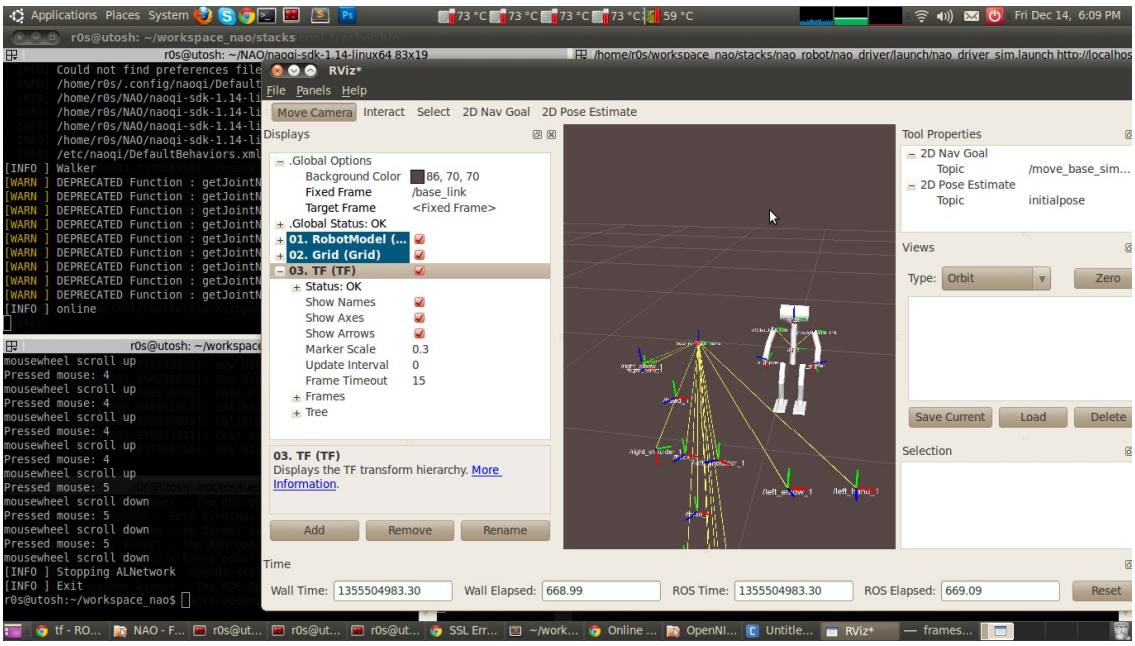

Illustration 15: Datos del Kinect y de Nao corriendo al mismo tiempo.

Adaptación dinámica de diferencia de estructuras entre humano y Nao

Dado que la relación entre el brazo y antebrazo de un humano y la del Nao no es la misma (ni entre dos humanos distintos) hay que hacer un escalado de los datos.
Para ello se calcula cada vez que hay un cambio de coordenadas en el esqueleto del usuario la escala correspondiente para adaptar los datos de la posición relativa de cada articulación con el pecho del usuario al tamaño del Nao.

Prueba de modelo de Nao en gazebo

A través del proyecto final de carrera de Jorge Bermejo Juárez del 2008-2009 se intentó correr la simulación de Nao en Gazebo (el simulador de físicas oficial de ROS).
Tras invertir unos días en ello esto no fue posible. Se intentó contactar con el autor pero nunca respondió.
Entre los problemas encontrados se encuentra una diferencia de versión de Gazebo y el uso de archivos 3D (de 3D Studio Max) no soportados en la versión de Gazebo de ROS.

Parte 3: Aplicación

Descripción

La aplicación a la que finalmente se quiere llegar con este proyecto es una aproximación con los medios de los que disponemos de lo que se puede ver en la película Acero Puro.
En esta película se teleopera un robot humanoide luchador imitando los movimientos de un exluchador de boxeo profesional.

Illustration 16: Cartel de la película Acero Puro.

Esta aplicación dista mucho de haberse llegado a cumplir.

Se ha conseguido implementar una solución para probar los distintos métodos de los que dispone el framework de Nao para mover el robot.

Otras posibles aplicaciones una vez finalizada una implementación del movimiento de un robot humanoide por imitación podrían ser:

-Juegos para gente (especialmente niños) que les animen a moverse. Esto es especialmente interesante para gente con movilidad limitada con posibilidad de recuperar parte de ella.

-Controlado de marionetas electrónicas. Por diferentes motivos puede resultar complicado teleoperar una marioneta con unos controles habituales, si esta marioneta tiene o se puede asemejar una estructura humanoide se podría controlar con una versión de un software como el que pretendemos crear. Esto podría solucionar problemas de tamaño en las marionetas. Puede haberlas demasiado pequeñas para que una persona esté dentro fingiendo el personaje o todo lo contrario, demasiado grandes para que una persona pueda moverlas.

-Juegos en general. Se puede tomar el enfoque de, por ejemplo, la consola Wii que consta de diferentes juegos imitando deportes, por ejemplo el golf donde se usa el mando como palo. Se podría dar a un robot determinado la herramienta necesaria y usar todo el cuerpo de un usuario para moverlo. Esto añadiría realismo al juego a la vez que originalidad.

Funcionamiento

La aplicación de prueba creada se ejecuta (una vez teniendo las librerías instaladas necesarias) simplemente ejecutando en un terminal:
move_arms_keyboard_setposition.py

Con las teclas direccionales del teclado y la rueda del ratón y (si se tienen) las teclas especiales del ratón se puede manejar la posición de la mano derecha del Nao.

Es necesario tener el robot Nao encendido, no importa su posición dado que al iniciar la aplicación se pondrá de pie de forma segura (y al cerrar la aplicación se sentará) y con la dirección IP por defecto. En caso contrario basta con abrir el archivo move_arms_keyboard_setposition.py y cambiar la IP a la del robot en el campo correspondiente.

Parte 4: Conclusiones / Lineas futuras de trabajo

Conclusión general de lo logrado

En este corto espacio de tiempo ha sido posible instalar y probar de forma básica los diferentes elementos que componen este proyecto.

Se han podido implementar algunas soluciones a problemas inherentes en este reto como:

-Escalado humano-robot.

-Integración Nao en ROS.

-Integración información de Nao y Kinect.

-Testeo de funciones disponibles en el framework de Nao.

Conclusión sobre viabilidad de uso de este robot

Este robot tiene una considerable limitación en CPU al ser el modelo antiguo de Nao. Esto se puede solucionar actualizandose a la última versión.
Aún así podría no ser suficiente para teleoperarlo a una velocidad real. La clave para el éxito en este caso sería hacer todos los cálculos intensivos de CPU en una máquina externa. Para esto hay que reimplementar la cinemática y dinámica del robot (o conseguir que Aldebaran Robotics nos dé el código fuente de la parte integrada en su framework).
Por lo demás es un robot ideal para esta tarea.

Otros detalles

El Kinect gracias a openni_tracker nos proporciona un esqueleto muy completo pero con una considerable cantidad de ruido. Este ruido es facilmente filtrable pero al filtrarlo se pierde cierta velocidad en la respuesta.
Para el objetivo principal de este proyecto, la lucha de robots teleoperados, es preferible que haya un pequeño margen de error pero que responda rápidamente.

Se ha utilizado la librería Tkinter para interactuar con el teclado y el ratón y ha sido sumamente sencillo integrarlo.

Se encontró otro proyecto donde Nick Pavlakis teleopera los brazos de un Nao a partir de los datos de un Kinect. En su proyecto se ve reflejado el problema de la velocidad de reacción a las ordenes dadas al robot tal como decribí anteriormente.

Lineas futuras de trabajo

Es necesario reimplementar la cinemática y dinámica del robot para computarla externamente. No es un problema nuevo, pero varía en cada robot. Son horas de trabajo.

Para obtener la dinámica del robot podemos leer el amperaje que está pasando por cada servo cada 10ms. Gracias a esto podemos aplicar una técnica llamada:

Compliant humanoid robot control by the torque transformer
La cual nos permite, a partir de la información del amperaje que pasa por cada servo dar ordenes de fuerza a través de ordenes de posición.

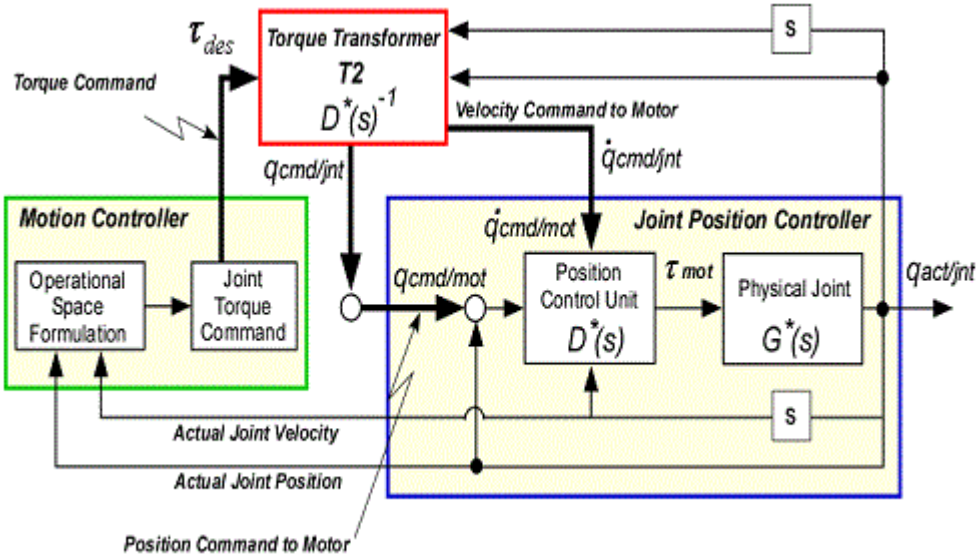

Illustration 17: Esquema de funcionamiento de la transformada de par de fuerzas.

En este esquema podemos observar como tenemos un bloque Motion Controller el cual se ocupa de, teniendo los datos de la dinámica del sistema, crea ordenes de fuerza que a través del Torque Transformer se convierten en ordenes de posición para que el Joint Position Controller pueda mandar ordenes a los servos (dado que los servos solo entienden de posición). Además se pueden dar ordenes de velocidad siguiendo el mismo sistema, con lo cual también ordenes de aceleración.

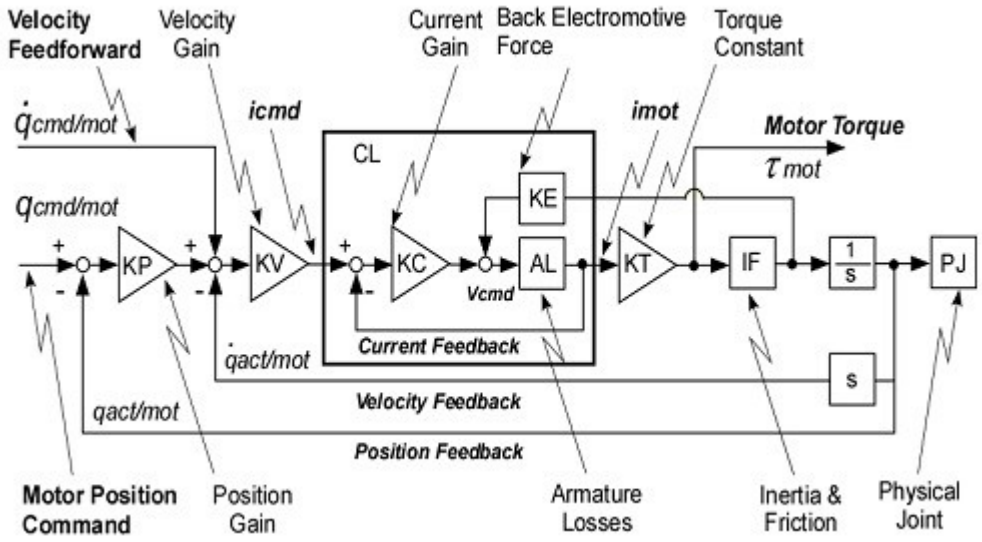

Illustration 18: Esquema de orden de posición que finalmente da orden de fuerza.

Este esquema extiende la idea anterior en un formato de bloques. Se ve explícitamente como el amperaje sirve para sacar el par de fuerzas actual en la caja “Current Feedback”.

Bibliografia

Sobre Wataru Yoshikazi
Paper: An Actuated Physical Puppet as an Input Device for Controlling a Digital Manikin
http://c15x4fug.securesites.net/projects/puppet/puppet.pdf
Web del proyecto V-sido
http://vsido.uijin.com/en/tech/tech.html

Documentación proyecto final de carrera Sammy Pfeiffer
http://upcommons.upc.edu/pfc/bitstream/2099.1/12454/1/74118.pdf

Sobre ROS
http://en.wikipedia.org/wiki/ROS_(Robot_Operating_System)
http://www.ros.org/wiki/

Sobre Kinect
http://en.wikipedia.org/wiki/Kinect

Sobre Nao
Documentación oficial
http://www.aldebaran-robotics.com/documentation/index.html
http://www.aldebaran-robotics.com/documentation/dev/python/examples.html#python-examples
http://www.aldebaran-robotics.com/documentation/family/nao_h25/index_h25.html
Tutorial Nao en ROS
http://www.ros.org/wiki/nao/Tutorials/Getting-Started

Sobre posibles aplicaciones de este sistema
A Computational Method for Physical Rehabilitation Assessment
http://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CDEQFjAA&url=http%3A%2F%2Fsmartech.gatech.edu%2Fjspui%2Fbitstream%2F1853%2F38301%2F2%2FIEEE_2010_BioRob_001.pdf&ei=noC4UO2EApLk8gTdv4HABw&usg=AFQjCNGOeDfEt86qD4taKa1Xv0XNOyu6Gg

Quantifying Upper-Arm Rehabilitation Metrics for Children through Interaction with a Humanoid Robot
http://iospress.metapress.com/content/f3525520235406h6/

Sobre comandos de fuerza a través de posición conociendo amperaje de los servos
Compliant humanoid robot control by the torque transformer

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5353907&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5353907

Explicación sobre Honda ASIMO:
https://sites.google.com/site/mindhumanoidrobotic/research-areas/motion-planning/torque-transformer

Texturas 3D modelo Nao
https://simspark.svn.sourceforge.net/svnroot/simspark/trunk/

Proyecto final de carrera Nao en gazebo
http://jderobot.org/index.php/Jbermejo-pfc-itis

Anexos

Repositorio opensource donde se puede encontrar el material del proyecto:

https://github.com/awesomebytes/nao_ros_testing

Código demo movimiento de brazo de Nao con ratón y teclado

```
#! /usr/bin/env python

# move arms with keyboard in X,Y axis

from __future__ import division
import sys
import time
from naoqi import ALProxy
import motion

import Tkinter as tk # for capturing keyboard


def moveEffector(effector):
  global motionProxy
  global postureProxy
  global x_
  global y_
  global z_
  global wx_
  global wy_
  global wz_

  positionChange = [x_, y_, z_, wx_, wy_, wz_]
  print "NewPos: " + str(positionChange)
  effectorName     = effector
  space            = motion.FRAME_ROBOT
  fractionMaxSpeed = 0.8
  axisMask       = 63

  motionProxy.post.setPosition(effectorName, space, positionChange,
fractionMaxSpeed, axisMask)


def mouse_wheelCallback(event):
  global motionProxy
  global postureProxy
  global x_
  global y_
  global z_
  global wx_
  global wy_
```

```
  global wz_

  print "Pressed mouse: " + str(event.num)
  # respond to Linux or Windows wheel event
  if event.num == 5 or event.delta == -120: # scroll down
    print "mousewheel scroll down"
    z_ = z_ - 0.02
    moveEffector("RArm")
  if event.num == 4 or event.delta == 120: # scroll up
    print "mousewheel scroll up"
    z_ = z_ + 0.02
    moveEffector("RArm")
  if event.num == 9: # lateral front mouse key
    wy_ = wy_ + 3.14/16
    moveEffector("Torso")
  if event.num == 8: # lateral back key
    wy_ = wy_ - 3.14/16
    moveEffector("Torso")

  if event.num == 6: # to the left scroll button
    wx_ = wx_ + 3.14/16
    moveEffector("Torso")
  if event.num == 7: # to the right scroll button
    wx_ = wx_ - 3.14/16
    moveEffector("Torso")


def keyCallback(event):
  global motionProxy
  global postureProxy
  global x_
  global y_
  global z_
  global wx_
  global wy_
  global wz_


  if event.keysym == 'Escape': # Exit if we press Esc, sit the robot
    postureProxy.goToPosture("Sit", 0.5)
    root.destroy()

  if event.char == event.keysym: # normal number and letter characters
    print( 'Normal Key %r' % event.char )
  elif len(event.char) == 1: # charcters like []/.,><#$ also Return and ctrl/key
    print( 'Punctuation Key %r (%r)' % (event.keysym, event.char) )
  else: # f1 to f12, shift keys, caps lock, Home, End, Delete ...
    print( 'Special Key %r' % event.keysym )
    if event.keysym == 'Up':
      x_ = x_ + 0.02
    elif event.keysym == 'Down':
      x_ = x_ - 0.02
    elif event.keysym == 'Left':
      y_ = y_ + 0.02
    elif event.keysym == 'Right':
      y_ = y_ - 0.02
```

```
    positionChange = [x_, y_, z_, wx_, wy_, wz_]
    print "NewPos: " + str(positionChange)
    #print "Move RArm 1"
    effectorName     = "RArm"
    space            = motion.FRAME_ROBOT
    fractionMaxSpeed = 0.8
    axisMask         = 7
    motionProxy.post.setPosition(effectorName, space, positionChange,
fractionMaxSpeed, axisMask)



#robotIP = "169.254.232.52"
robotIP = "127.0.0.1"
PORT = 9559
global motionProxy
global postureProxy
global x_
global y_
global z_
global wx_
global wy_
global wz_

x_ = y_ = z_ = wx_ = wy_ = wz_ = 0.0
try:
  motionProxy = ALProxy("ALMotion", robotIP, PORT)
except Exception,e:
  print "Could not create proxy to ALMotion"
  print "Error was: ",e
  sys.exit(1)

try:
  postureProxy = ALProxy("ALRobotPosture", robotIP, PORT)
except Exception, e:
  print "Could not create proxy to ALRobotPosture"
  print "Error was: ", e

postureProxy.goToPosture("StandInit", 0.5)



root = tk.Tk()
print( "Press a key (Escape key to exit):" )
root.bind('<KeyPress>', keyCallback)
# with Windows OS
root.bind("<MouseWheel>", mouse_wheelCallback)
# with Linux OS
root.bind("<ButtonPress>", mouse_wheelCallback)

print "Press Up/Down arrow keys to Move in X"
print "Press Left/Right to move in Y"
print "Use mousewheel scroll up/down to move in Z (over the window)"
```

```
root.mainloop()
```

Tutorial movimiento cartesiano Nao:

http://www.aldebaran-robotics.com/documentation/dev/python/examples/motion/cartesian.html#python-example-motion-cartesian

```
# -*- encoding: UTF-8 -*-

'''Cartesian control: Multiple Effector Trajectories'''

import sys
import motion
import almath
from naoqi import ALProxy


def StiffnessOn(proxy):
    # We use the "Body" name to signify the collection of all joints
    pNames = "Body"
    pStiffnessLists = 1.0
    pTimeLists = 1.0
    proxy.stiffnessInterpolation(pNames, pStiffnessLists, pTimeLists)


def main(robotIP):
    ''' Move the torso and keep arms fixed in nao space
    Warning: Needs a PoseInit before executing
    '''

    # Init proxies.
    try:
        motionProxy = ALProxy("ALMotion", robotIP, 9559)
    except Exception, e:
        print "Could not create proxy to ALMotion"
        print "Error was: ", e

    try:
        postureProxy = ALProxy("ALRobotPosture", robotIP, 9559)
    except Exception, e:
        print "Could not create proxy to ALRobotPosture"
        print "Error was: ", e

    # Set NAO in Stiffness On
    StiffnessOn(motionProxy)

    # Send NAO to Pose Init
    postureProxy.goToPosture("StandInit", 0.5)

    space      = motion.FRAME_ROBOT
    isAbsolute = False

    effectorList = ["LArm", "RArm"]

    # Motion of Arms with block process
    axisMaskList = [almath.AXIS_MASK_VEL, almath.AXIS_MASK_VEL]
```

```
    timesList    = [[1.0], [1.0]]          # seconds
    pathList     = [[[0.0, -0.04, 0.0, 0.0, 0.0, 0.0]],
                    [[0.0,  0.04, 0.0, 0.0, 0.0, 0.0]]]
    motionProxy.positionInterpolations(effectorList, space, pathList,
                                       axisMaskList, timesList, isAbsolute)


    effectorList = ["LArm", "RArm", "Torso"]

    # Motion of Arms and Torso with block process
    axisMaskList = [almath.AXIS_MASK_VEL,
                    almath.AXIS_MASK_VEL,
                    almath.AXIS_MASK_ALL]

    timesList    = [[4.0],                  # LArm  in seconds
                    [4.0],                    # RArm  in seconds
                    [1.0, 2.0, 3.0, 4.0]]   # Torso in seconds

    dx           = 0.03                   # translation axis X (meters)
    dy           = 0.04                   # translation axis Y (meters)

    pathList     = [[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], # LArm do not move
                    [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], # RArm do not move
                    [[0.0, +dy, 0.0, 0.0, 0.0, 0.0],  # Torso point 1
                     [-dx, 0.0, 0.0, 0.0, 0.0, 0.0],  # Torso point 2
                     [0.0, -dy, 0.0, 0.0, 0.0, 0.0],  # Torso point 3
                     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]  # Torso point 4
                    ]
    motionProxy.positionInterpolations(effectorList, space, pathList,
                                       axisMaskList, timesList, isAbsolute)


if __name__ == "__main__":
    robotIp = "127.0.0.1"

    if len(sys.argv) <= 1:
        print "Usage python motion_cartesianTorsoArm2.py robotIP (optional default:
127.0.0.1)"
    else:
        robotIp = sys.argv[1]

    main(robotIp)
```

Tutorial "whole body motion" Nao:

http://www.aldebaran-robotics.com/documentation/dev/python/examples/motion/whole_body.html#python-example-motion-wholebody

```
# -*- encoding: UTF-8 -*-

''' Whole Body Motion: Multiple Effectors control '''

import sys
import motion
import almath
import time
from naoqi import ALProxy


def StiffnessOn(proxy):
    # We use the "Body" name to signify the collection of all joints
    pNames = "Body"
    pStiffnessLists = 1.0
    pTimeLists = 1.0
    proxy.stiffnessInterpolation(pNames, pStiffnessLists, pTimeLists)


def main(robotIP):
    '''
        Example of a whole body multiple effectors control "LArm", "RArm" and "Torso"
        Warning: Needs a PoseInit before executing
                 Whole body balancer must be inactivated at the end of the script
    '''

    # Init proxies.
    try:
        proxy = ALProxy("ALMotion", robotIP, 9559)
    except Exception, e:
        print "Could not create proxy to ALMotion"
        print "Error was: ", e

    try:
        postureProxy = ALProxy("ALRobotPosture", robotIP, 9559)
    except Exception, e:
        print "Could not create proxy to ALRobotPosture"
        print "Error was: ", e

    # Set NAO in Stiffness On
    StiffnessOn(proxy)

    # Send NAO to Pose Init
    postureProxy.goToPosture("StandInit", 0.5)

    # Enable Whole Body Balancer
    isEnabled  = True
    proxy.wbEnable(isEnabled)

    # Legs are constrained fixed
```

```
stateName  = "Fixed"
supportLeg = "Legs"
proxy.wbFootState(stateName, supportLeg)

# Constraint Balance Motion
isEnable   = True
supportLeg = "Legs"
proxy.wbEnableBalanceConstraint(isEnable, supportLeg)

# Arms motion
effectorList = ["LArm", "RArm"]

space        = motion.FRAME_ROBOT

pathList     = [
                  [
                   [0.0,   0.08,  0.14, 0.0, 0.0, 0.0], # target 1 for "LArm"
                   [0.0,  -0.05, -0.07, 0.0, 0.0, 0.0], # target 2 for "LArm"
                   [0.0,   0.08,  0.14, 0.0, 0.0, 0.0], # target 3 for "LArm"
                   [0.0,  -0.05, -0.07, 0.0, 0.0, 0.0], # target 4 for "LArm"
                   [0.0,   0.08,  0.14, 0.0, 0.0, 0.0], # target 5 for "LArm"
                   ],
                  [
                   [0.0,   0.05, -0.07, 0.0, 0.0, 0.0], # target 1 for "RArm"
                   [0.0,  -0.08,  0.14, 0.0, 0.0, 0.0], # target 2 for "RArm"
                   [0.0,   0.05, -0.07, 0.0, 0.0, 0.0], # target 3 for "RArm"
                   [0.0,  -0.08,  0.14, 0.0, 0.0, 0.0], # target 4 for "RArm"
                   [0.0,   0.05, -0.07, 0.0, 0.0, 0.0], # target 5 for "RArm"
                   [0.0,  -0.08,  0.14, 0.0, 0.0, 0.0], # target 6 for "RArm"
                   ]
                  ]

axisMaskList = [almath.AXIS_MASK_VEL, # for "LArm"
                  almath.AXIS_MASK_VEL] # for "RArm"

coef       = 1.5
timesList  = [ [coef*(i+1) for i in range(5)],  # for "LArm" in seconds
                 [coef*(i+1) for i in range(6)] ] # for "RArm" in seconds

isAbsolute   = False

# called cartesian interpolation
proxy.positionInterpolations(effectorList, space, pathList,
                             axisMaskList, timesList, isAbsolute)

# Torso Motion
effectorList = ["Torso", "LArm", "RArm"]

dy = 0.06
dz = 0.06
pathList     = [
                  [
                   [0.0, +dy, -dz, 0.0, 0.0, 0.0], # target  1 for "Torso"
                   [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target  2 for "Torso"
                   [0.0, -dy, -dz, 0.0, 0.0, 0.0], # target  3 for "Torso"
                   [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target  4 for "Torso"
```

```
                      [0.0, +dy, -dz, 0.0, 0.0, 0.0], # target  5 for "Torso"
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target  6 for "Torso"
                      [0.0, -dy, -dz, 0.0, 0.0, 0.0], # target  7 for "Torso"
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target  8 for "Torso"
                      [0.0, +dy, -dz, 0.0, 0.0, 0.0], # target  9 for "Torso"
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target 10 for "Torso"
                      [0.0, -dy, -dz, 0.0, 0.0, 0.0], # target 11 for "Torso"
                      [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], # target 12 for "Torso"
                      ],
                      [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], # for "LArm"
                      [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], # for "LArm"
                     ]

    axisMaskList = [almath.AXIS_MASK_ALL, # for "Torso"
                    almath.AXIS_MASK_VEL, # for "LArm"
                    almath.AXIS_MASK_VEL] # for "RArm"

    coef       = 0.5
    timesList  = [
                   [coef*(i+1) for i in range(12)], # for "Torso" in seconds
                   [coef*12],                       # for "LArm" in seconds
                   [coef*12]                        # for "RArm" in seconds
                  ]

    isAbsolute   = False

    proxy.positionInterpolations(effectorList, space, pathList,
                                 axisMaskList, timesList, isAbsolute)


    # Deactivate whole body
    isEnabled    = False
    proxy.wbEnable(isEnabled)

    # Send NAO to Pose Init
    postureProxy.goToPosture("StandInit", 0.5)


if __name__ == "__main__":
    robotIp = "127.0.0.1"

    if len(sys.argv) <= 1:
        print "Usage python motion_wbMultipleEffectors.py robotIP (optional default:
127.0.0.1)"
    else:
        robotIp = sys.argv[1]

    main(robotIp)
```

Anexo urdf nao en nao_common

nao_robot_v3_3d.urdf.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro" name="nao">
<!--
  ROS urdf definition of the Nao humanoid robot by Aldebaran Robotics

  This is part of the Nao-Stack of University of Freiburg, available at:
  http://code.google.com/p/alufr-ros-pkg/

  Authors: Armin Hornung, Stefan Osswald

  Joint names and properties are according to the Nao-documentation of
  Aldebaran Robotics ("Kinematics 3.2" for Nao V3.2).

  Note that according to the documentation, the joint limits for the left leg
  are different from the corresponding limits for the right leg.
  For {L,R}HipPitch, {L,R}KneePitch and {L,R}AnklePitch the smallest interval
  of the joint limits was chosen, whereas for {L,R}HipRoll and {L,R}AnkleRoll
  the respective (differing) limits were copied from the documentation.

  The joint limits for {L,R}ShoulderRoll and {L,R}ElbowRoll do
  not include the zero position.

  This file loads a basic visualization of Nao and the structure (you need both)

-->

  <include filename="$(find nao_description)/urdf/visuals_3d.xacro" />
  <include filename="$(find
nao_description)/urdf/nao_robot_v3_structure.urdf.xacro" />


</robot>
```

nao_robot_v3_structure.urdf.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
<!--
  ROS urdf definition of the Nao humanoid robot by Aldebaran Robotics

  DO NOT INCLUDE THIS FILE DIRECTLY! USE nao_robot.urdf.xacro INSTEAD!

  This is part of the Nao-Stack of University of Freiburg, available at:
  http://code.google.com/p/alufr-ros-pkg/

  Authors: Armin Hornung, Stefan Osswald

  Joint names and properties are according to the Nao-documentation of
  Aldebaran Robotics ("Kinematics 3.2" for Nao V3.2) and REP-120:
  http://ros.org/reps/rep-0120.html

  Note that according to the documentation, the joint limits for the left leg
  are different from the corresponding limits for the right leg.
  For {L,R}HipPitch, {L,R}KneePitch and {L,R}AnklePitch the smallest interval
  of the joint limits was chosen, whereas for {L,R}HipRoll and {L,R}AnkleRoll
  the respective (differing) limits were copied from the documentation.

  The joint limits for {L,R}ShoulderRoll and {L,R}ElbowRoll do
  not include the zero position.

  TODO: effort and velocity limits are DUMMY values!
  TODO: Inertial matrix is wrong or contains dummdy values.
        Docs say inertial matrix is in link frame, URDF expects CoM frame
-->

  <!-- dummy first link, directly connects with torso -->
  <link name="base_link" />

  <!-- first real link of the robot, inertial reference frame -->
  <link name="torso" >
  <!--<inertial>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <mass value="0"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial> -->
    <inertial>
      <origin xyz="-0.00480 0.00006 0.04227" rpy="0 0 0"/>
      <mass value="1.02628"/>
      <inertia ixx="0.00487953284"  ixy="-0.00001428591"  ixz="-0.00019545651"
iyy="0.00470360698" iyz="0.00002224589" izz="0.0015671352" />
    </inertial>
    <xacro:insert_visualization_torso/>
  </link>

  <link name="HeadYaw_link">
    <inertial>
      <origin xyz="-0.00003 0.00018 -0.02573" rpy="0 0 0"/>
      <mass value="0.05959"/>
```

```
      <inertia ixx="0.00006223368"  ixy="0.00000000042"  ixz="0.00000007448"
iyy="0.00006324431" iyz="0.00000010416" izz="0.00000549462" />
    </inertial>
    <xacro:insert_visualization_headYaw/>
  </link>

  <link name="HeadPitch_link">
    <inertial>
      <origin xyz="0.00383 -0.00093 0.05156" rpy="0 0 0"/>
      <mass value="0.47671"/>
      <inertia ixx="0.00208895741"  ixy="0.00000549094"  ixz="0.0001133995"
iyy="1932222.63e-9" iyz="-0.00002803917" izz="0.00082257793" />
    </inertial>
    <xacro:insert_visualization_head/>
  </link>

  <link name="gaze" />

  <link name="CameraTop_frame">
    <xacro:insert_visualization_cameraTop/>
  </link>

  <link name="CameraBottom_frame">
    <xacro:insert_visualization_cameraBottom/>
  </link>

  <!-- Arm -->
  <xacro:macro name="arm_links" params="side reflect">
  <link name = "${side}ShoulderPitch_link">
    <inertial>
      <origin xyz="-0.00178 ${reflect*0.02507} 0.00019" rpy="0 0 0"/>
      <mass value="0.06984"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
  </link>

  <link name = "${side}ShoulderRoll_link" >
    <inertial>
      <origin xyz="0.02067 ${reflect*-0.00388} 0.00362" rpy="0 0 0"/>
      <mass value="0.12166"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
    <xacro:insert_visualization_arm side="${side}" reflect="${reflect}"/>
  </link>

  <link name = "${side}ElbowYaw_link">
    <inertial>
      <origin xyz="-0.02573 ${reflect*0.00001} -0.00020" rpy="0 0 0"/>
      <mass value="0.05959"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>

    <xacro:insert_visualization_elbow_yaw side="${side}"/>
  </link>

  <link name = "${side}ElbowRoll_link">
```

```
  <inertial>
    <origin xyz="0.01940 ${reflect*-0.00304} 0.00250" rpy="0 0 0"/>
    <mass value="0.03770"/>
    <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
  </inertial>

  <xacro:insert_visualization_elbow side="${side}" reflect="${reflect}"/>
</link>

<link name = "${side}WristYaw_link">
  <inertial>
    <origin xyz="0.03241 ${reflect*-0.00268} -0.00272" rpy="0 0 0"/>
    <mass value="0.14314"/>
    <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
  </inertial>
  <xacro:insert_visualization_wrist side="${side}" reflect="${reflect}"/>
</link>
</xacro:macro>
<link name="l_wrist"/>
<link name="r_wrist"/>

<link name="l_gripper">
  <xacro:insert_visualization_fingers side="L"/>
</link>
<link name="r_gripper">
  <xacro:insert_visualization_fingers side="R"/>
</link>

<xacro:arm_links side="L" reflect="-1"/>
<xacro:arm_links side="R" reflect="1"/>

<!-- Leg links -->
<xacro:macro name="leg_links" params="side reflect">
<link name="${side}HipYawPitch_link" >
  <inertial>
    <origin xyz="-0.00717 ${reflect*0.01187} 0.02705" rpy="0 0 0"/>
    <mass value="0.07244"/>
    <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
  </inertial>
  <xacro:insert_visualization_hip side="${side}"/>
      </link>


<link name="${side}HipRoll_link">
  <inertial>
    <origin xyz="-0.01649 ${reflect*-0.00029} -0.00475" rpy="0 0 0"/>
    <mass value="0.13530"/>
    <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
  </inertial>
  <xacro:insert_visualization_hipRoll side="${side}"/>
</link>

<link name="${side}HipPitch_link">
  <inertial>
    <origin xyz="0.00131 ${reflect*-0.00201} -0.05386" rpy="0 0 0"/>
    <mass value="0.39798"/>
```

```
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
    <xacro:insert_visualization_thigh side="${side}" reflect="${reflect}"/>
  </link>

  <link name="${side}KneePitch_link">
    <inertial>
      <origin xyz="0.00471 ${reflect*-0.00210} -0.04891" rpy="0 0 0"/>
      <mass value="0.29706"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
    <xacro:insert_visualization_shine side="${side}" reflect="${reflect}"/>
  </link>

  <link name="${side}AnklePitch_link">
    <inertial>
      <origin xyz="0.00142 ${reflect*-0.00028} 0.00638" rpy="0 0 0"/>
      <mass value="0.13892"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
    <xacro:insert_visualization_ankle side="${side}"/>
  </link>

  <link name="${side}AnkleRoll_link">
   <inertial>
      <origin xyz="0.02489 ${reflect*-0.00330} -0.03208" rpy="0 0 0"/>
      <mass value="0.16304"/>
      <inertia ixx="0"  ixy="0"  ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
    <xacro:insert_visualization_ankleRoll side="${side}"/>
  </link>
  </xacro:macro>
  <link name="l_ankle" />
  <link name="r_ankle" />

  <link name="l_sole">
    <xacro:insert_visualization_foot side="L"/>
  </link>
  <link name="r_sole">
    <xacro:insert_visualization_foot side="R"/>
  </link>

  <xacro:leg_links side="L" reflect="-1"/>
  <xacro:leg_links side="R" reflect="1"/>

<!--   Joints following below -->

  <joint name="base_joint" type="fixed">
    <child link="torso"/>
    <parent link="base_link"/>
    <origin xyz="0 0 0" rpy="0 0 0" />
  </joint>

  <!-- Head joints -->

  <joint name="HeadYaw" type="revolute">
```

```
  <parent link="torso"/>
  <child link="HeadYaw_link"/>
      <!-- NeckOffsetZ -->
  <origin xyz="0 0 0.1265" rpy="0 0 0" />
  <axis xyz="0 0 1" />
  <limit velocity="3.0" effort="30" lower="-2.0857" upper="2.0857" />
</joint>

<joint name="HeadPitch" type="revolute">
  <parent link="HeadYaw_link"/>
  <child link="HeadPitch_link"/>
  <origin xyz="0 0 0" rpy="0 0 0" />
  <axis xyz="0 1 0" />
  <limit velocity="3.0" effort="30" lower="-0.6720" upper="0.5149" />
</joint>

<joint name="gaze_joint" type="fixed">
  <parent link="HeadPitch_link"/>
  <child link="gaze"/>
  <origin xyz="0.0539 0 0.05" />
</joint>

<joint name="CameraTop" type="fixed">
  <parent link="HeadPitch_link"/>
  <child link="CameraTop_frame"/>
  <origin xyz="0.0539 0 0.0679" rpy="-${pi_2} 0 -${pi_2}" />
</joint>

<joint name="CameraBottom" type="fixed">
  <parent link="HeadPitch_link"/>
  <child link="CameraBottom_frame"/>
  <!-- rotated by 40 degrees (pitch) -->
  <origin xyz="0.0488 0 0.02381" rpy="-2.26893 0 -${pi_2}" />
</joint>


<!-- Arm joints -->
<xacro:macro name="arm_joints" params="side reflect">
<joint name="${side}ShoulderPitch" type="revolute">
  <parent link="torso"/>
  <child link="${side}ShoulderPitch_link"/>
  <origin xyz="0 ${reflect*-0.098} 0.1" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <limit velocity="3.0" effort="30" lower="-2.0857" upper="2.0857"/>
</joint>

<joint name="${side}ElbowYaw" type="revolute">
  <parent link="${side}ShoulderRoll_link"/>
  <child link="${side}ElbowYaw_link"/>
  <origin xyz="0.09 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-2.0857" upper="2.0857"/>
</joint>


<joint name="${side}WristYaw" type="revolute">
```

```
  <parent link="${side}ElbowRoll_link"/>
  <child link="${side}WristYaw_link"/>
  <origin xyz="0.05055 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-1.8238" upper="1.8238"/>
</joint>
</xacro:macro>

<joint name="l_wrist_joint" type="fixed">
  <parent link="LWristYaw_link" />
  <child link="l_wrist" />
  <origin xyz="0 0 0" />
</joint>
<joint name="r_wrist_joint" type="fixed">
  <parent link="RWristYaw_link" />
  <child link="r_wrist" />
  <origin xyz="0 0 0" />
</joint>

<joint name="l_gripper_joint" type="fixed">
  <parent link="l_wrist" />
  <child link="l_gripper" />
  <origin xyz="0.058 0 -0.0159" />
</joint>
<joint name="r_gripper_joint" type="fixed">
  <parent link="r_wrist" />
  <child link="r_gripper" />
  <origin xyz="0.058 0 -0.0159" />
</joint>


<joint name="LElbowRoll" type="revolute">
  <parent link="LElbowYaw_link"/>
  <child link="LElbowRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit velocity="3.0" effort="30" lower="-1.5621" upper="-0.0087"/>
</joint>
<joint name="RElbowRoll" type="revolute">
  <parent link="RElbowYaw_link"/>
  <child link="RElbowRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit velocity="3.0" effort="30" lower="0.0087" upper="1.5621"/>
</joint>

<joint name="LShoulderRoll" type="revolute">
  <parent link="LShoulderPitch_link"/>
  <child link="LShoulderRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit velocity="3.0" effort="30" lower="0.0087" upper="1.6494"/>
</joint>
<joint name="RShoulderRoll" type="revolute">
  <parent link="RShoulderPitch_link"/>
  <child link="RShoulderRoll_link"/>
```

```
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit velocity="3.0" effort="30" lower="-1.6494" upper="-0.0087"/>
</joint>

<xacro:arm_joints side="L" reflect="-1"/>
<xacro:arm_joints side="R" reflect="1"/>

<!-- Leg joints -->
<joint name="LHipYawPitch" type="revolute">
  <parent link="torso"/>
  <child link="LHipYawPitch_link"/>
  <origin xyz="0 0.05 -0.085" rpy="0 0 0"/>
  <axis xyz="0 0.707106 -0.707106"/>
  <limit velocity="3.0" effort="30" lower="-1.145303" upper="0.740810"/>
</joint>
<joint name="RHipYawPitch" type="revolute">
  <parent link="torso"/>
  <child link="RHipYawPitch_link"/>
  <origin xyz="0 -0.05 -0.085" rpy="0 0 0"/>
  <axis xyz="0 0.707106 0.707106"/>
  <limit velocity="3.0" effort="30" lower="-1.145303" upper="0.740810"/>
  <mimic joint="LHipYawPitch" multiplier="1" offset="0"/>
</joint>

<xacro:macro name="leg_joints" params="side reflect">
<joint name="${side}HipPitch" type="revolute">
  <parent link="${side}HipRoll_link"/>
  <child link="${side}HipPitch_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <!-- LHipPitch limits: -1.773912 .. 0.484090 -->
  <!-- RHipPitch limits: -1.772308 .. 0.485624 -->
  <limit velocity="3.0" effort="30" lower="-1.772308" upper="0.484090"/>
</joint>

<joint name="${side}KneePitch" type="revolute">
  <parent link="${side}HipPitch_link"/>
  <child link="${side}KneePitch_link"/>
  <origin xyz="0 0 -0.1" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <!-- LKneePitch limits: -0.092346 .. 2.112528 -->
  <!-- RKneePitch limits: -0.103083 .. 2.120198 -->
  <limit velocity="3.0" effort="30" lower="-0.092346" upper="2.112528"/>
</joint>

<joint name="${side}AnklePitch" type="revolute">
  <parent link="${side}KneePitch_link"/>
  <child link="${side}AnklePitch_link"/>
  <origin xyz="0 0 -0.10275" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <!-- LAnklePitch limits: -1.189516 .. 0.922747 -->
  <!-- RAnklePitch limits: -1.186448 .. 0.932056 -->
  <limit velocity="3.0" effort="30" lower="-1.186448" upper="0.922747"/>
</joint>
</xacro:macro>
```

```
<joint name="LHipRoll" type="revolute">
  <parent link="LHipYawPitch_link"/>
  <child link="LHipRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-0.379472" upper="0.790477"/>
</joint>
<joint name="RHipRoll" type="revolute">
  <parent link="RHipYawPitch_link"/>
  <child link="RHipRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-0.738321" upper="0.414754"/>
</joint>

<joint name="LAnkleRoll" type="revolute">
  <parent link="LAnklePitch_link"/>
  <child link="LAnkleRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-0.769001" upper="0.397880"/>
</joint>
<joint name="RAnkleRoll" type="revolute">
  <parent link="RAnklePitch_link"/>
  <child link="RAnkleRoll_link"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="1 0 0"/>
  <limit velocity="3.0" effort="30" lower="-0.388676" upper="0.785875"/>
</joint>

<joint name="l_ankle_joint" type="fixed">
  <parent link="LAnkleRoll_link" />
  <child link="l_ankle" />
  <origin xyz="0 0 0" />
</joint>
<joint name="r_ankle_joint" type="fixed">
  <parent link="RAnkleRoll_link" />
  <child link="r_ankle" />
  <origin xyz="0 0 0" />
</joint>
<joint name="l_sole_joint" type="fixed">
  <parent link="l_ankle" />
  <child link="l_sole" />
  <origin xyz="0 0 -0.04511" />
</joint>
<joint name="r_sole_joint" type="fixed">
  <parent link="r_ankle" />
  <child link="r_sole" />
  <origin xyz="0 0 -0.04511" />
</joint>


<xacro:leg_joints side="L" reflect="-1"/>
<xacro:leg_joints side="R" reflect="1"/>
```

```
</robot>
```

visuals_3d.xacro

```
<?xml version="1.0"?>
<!--
    A simple visualization of the Nao's kinematic structure as cylinders
-->
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <material name="LightGrey">
    <color rgba="0.8 0.8 0.8 1.0"/>
  </material>
  <material name="Grey">
      <color rgba="0.6 0.6 0.6 1.0"/>
  </material>
  <material name="DarkGrey">
      <color rgba="0.3 0.3 0.3 1.0"/>
  </material>
  <material name="Black">
      <color rgba="0.1 0.1 0.1 1.0"/>
  </material>

    <xacro:property name="pi" value="3.1415926535897931"/>
    <xacro:property name="pi_2" value="1.5707963267948966"/>

    <xacro:macro name="insert_visualization_torso" params="">
        <visual>
            <!-- one cylinder spanning over HipOffsetZ and NeckOffsetZ -->
            <origin xyz="0 0 0.02075" rpy="0 0 ${pi_2}"/>
            <geometry>
                <!-- <cylinder radius="0.015" length="0.2115"/> -->
                        <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/cuerpo.dae" scale=".1 .1 .1"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_headYaw" params="">
    </xacro:macro>

    <xacro:macro name="insert_visualization_head" params="">
        <visual>
            <origin xyz="0 0 0.058" rpy="1.57 0 -${pi_2}"/>
            <geometry>
                <!-- <cylinder radius="0.04" length="0.14"/> -->
                <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/head.dae" scale=".1 .1 .1"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_arm" params="side reflect">
        <visual>
```

```
            <origin xyz="0.045 0 0" rpy="${pi_2 * reflect} 0 ${pi_2 * reflect}"/>
            <geometry>
                <!--<cylinder radius="0.015" length="0.09"/> -->
                <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/UpperArm.dae" scale=".1 .1 .1"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_elbow" params="side reflect">
        <visual>
            <origin xyz="0.025325 0 0" rpy="${pi_2} 0 ${pi_2}"/>
            <geometry>
                <cylinder radius="0.015" length="0.05065"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_wrist" params="side reflect">
        <visual>
            <origin xyz="0.029 0 0" rpy="${pi_2 * reflect} ${pi * ((1 +
reflect)/2)} ${pi_2 * reflect}"/>
            <geometry>
                <!-- <cylinder radius="0.015" length="0.058"/> -->
                <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/LowerArm.dae" scale=".1 .1 .1"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_hip" params="side">
    </xacro:macro>

    <xacro:macro name="insert_visualization_hipRoll" params="side">
    </xacro:macro>

    <xacro:macro name="insert_visualization_thigh" params="side reflect">
        <visual>
            <origin xyz="0 0 -0.05" rpy="0 0 ${pi * ((1 + reflect)/2)}"/>
            <geometry>
                <!-- <cylinder radius="0.015" length="0.1"/> -->
        <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/thig.dae" scale=".05 .05 .05"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_shine" params="side reflect">
        <visual>
```

```
            <origin xyz="0 0 -0.05" rpy="${pi_2 } 0 ${pi_2 * reflect}"/>
            <geometry>
                <!-- <cylinder radius="0.015" length="0.1"/> -->
                <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/shank.dae" scale=".05 .05 .05"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_ankle" params="side">
    </xacro:macro>

    <xacro:macro name="insert_visualization_ankleRoll" params="side">
        <visual>
            <origin xyz="0 0 -0.023" rpy="0 0 0"/>
            <geometry>
                <cylinder radius="0.015" length="0.046"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_foot" params="side">
        <visual>
            <origin xyz="0.02 0 0.0075" rpy="${pi_2} ${pi} ${pi_2}"/>
            <geometry>
                <!-- <box size="0.16 0.06 0.015"/> -->
                <mesh
filename="file:///home/r0s/workspace_nao/gazebonao/gnao/bodyparts/blender_bodypart
s/dae_bodyparts/foot.dae" scale=".05 .05 .05"/>
            </geometry>
            <material name="LightGrey"/>
        </visual>
    </xacro:macro>

    <xacro:macro name="insert_visualization_cameraTop" params="">
    </xacro:macro>

    <xacro:macro name="insert_visualization_cameraBottom" params="">
    </xacro:macro>

    <xacro:macro name="insert_visualization_fingers" params="side">
    </xacro:macro>

    <xacro:macro name="insert_visualization_elbow_yaw" params="side">
    </xacro:macro>
</robot>
```