PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Facultad de Ciencias e Ingeniería

Módulo de Sensor 3D Externo para Robots Humanoides y su Aplicación para el mejoramiento de la interacción humano robot

Anexos

Tesis para optar el Título de Ingeniero Mecatrónico,

que presenta el bachiller:

José Alexander López Manrique

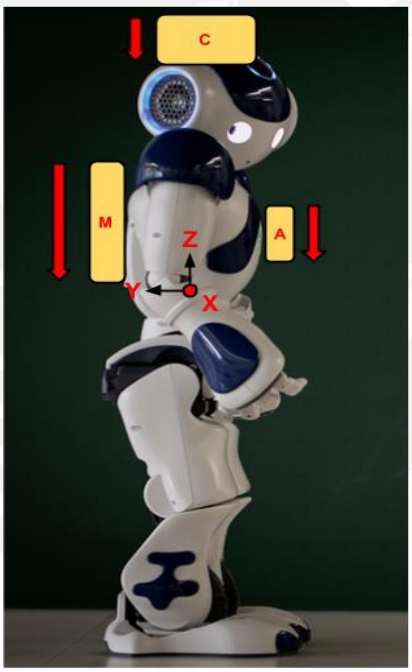
ASESOR: Christian Isaac Peñaloza Sánchez

Lima, febrero del 2016

Anexo A: Análisis y Cálculos del Sistema Mecatrónico

Cálculo por peso y estabilidad:

El centro de masa se puede calcular teniendo un valor aproximado del peso de la mochila hecha de PLA (plástico polímero usando en las impresiones 3D), se eligió este material ya que es el adecuado para la construcción de prototipos. Las medidas de las piezas determinan el volumen de estas, usando el valor de la densidad del material (PLA 1,050 g/cm^3) se puede calcular el peso de cada una de las partes del módulo. En capítulos previos se mencionó que el módulo del robot posee tres partes: La acopla sobre la cabeza del robot (parte C), la estructura que va en la espalda (parte M) y por último la estructura que va en el pecho (parte A); con la ubicación de estos con respecto al centro de gravedad, se puede determinar si el peso se encuentra equilibrado.



Partes	Peso (incluido los componentes)	Distancia al centro de gravedad (relativo al del robot)
Parte C	411.8 g	X: 0.78 mm Z: 279.11 mm Y: 11.25 mm
Parte M	157.5 g	X: 0.64 mm Z: 39.74 mm Y: 70.21 mm
Parte A	181.9 g	X: -3.07 mm Z: 48.55 mm Y: -70.09 mm
Cordón Superior D.	16.5 g	X: 31.5 mm Z: 106.21 mm Y: -5.03 mm

Cordón Superior I.	16.5 g	X: -31.5 mm Z: 106.21 mm Y: -5.03 mm
Cordón Inferior D.	5.5 g	X: 60.81 mm Z: 30.5 mm Y: -4.70 mm
Cordón Inferior I.	5.5 g	X: -60.81 mm Z: 30.5 mm Y: -4.70 mm

Momentos en el eje XZ:

M = Momento que realiza el robot para mantenerse en equilibrio.

$$411.8*11.25 + 123.5*70.21 = M + 181.9*70.09 + 16.5*(5.03 + 5.03) + 5.5*(4.7 + 4.7)$$

$$13303.685 = M + 12967.061$$

$$M = 336.624 \text{ g.mm}$$

Las especificaciones no dan a conocer si el robot puede soportar dicho torque sobre su cabeza por tanto se realizó una prueba de concepto, en la cual se determinó que el robot puede moverse o pararse aun con el peso del sistema sobre él.

Cálculos de resistencia:

La segunda y tercera parte del módulo se une al robot a través de soportes, los soportes superiores se colocan sobre los hombros y en sus extremos se sujeta a las partes de los módulos usando una pequeña interferencia geométrica o lengüeta. Estos elementos son los más críticos ya que tienen que cargar con la segunda y tercera parte del módulo, además de mantenerlos en equilibrio. En este caso el esfuerzo crítico es el que realiza la fuerza de corte en las zonas que se muestra en la figura 1.1, según la simulación realizada indicando que los puntos de apoyo se ubican sobre las interferencias geométricas de los soportes y este se apoya en el punto medio del soporte sobre los hombros del humanoide. Los resultados del análisis demuestran que el esfuerzo máximo es de 1.706 MPa, teniendo en cuenta que el esfuerzo máximo es de corte, se compara con el esfuerzo de corte máximo que puede soportar el PLA, el módulo de corte de este es de 2.4 GPa, dicho valor es muy superior al posible esfuerzo máximo que puede actuar en el soporte.

La fuerza sobre la lengüeta o interferencia geométrica ocasiona un desplazamiento de esta en 0,004429 mm, dicho desplazamiento es muy pequeño y puede ser despreciado debido a la función que tiene la lengüeta de soportar a las partes del módulo, dichas partes no tiene que tener una posición exacta con respecto a su posición sobre el robot para tener un correcto funcionamiento.

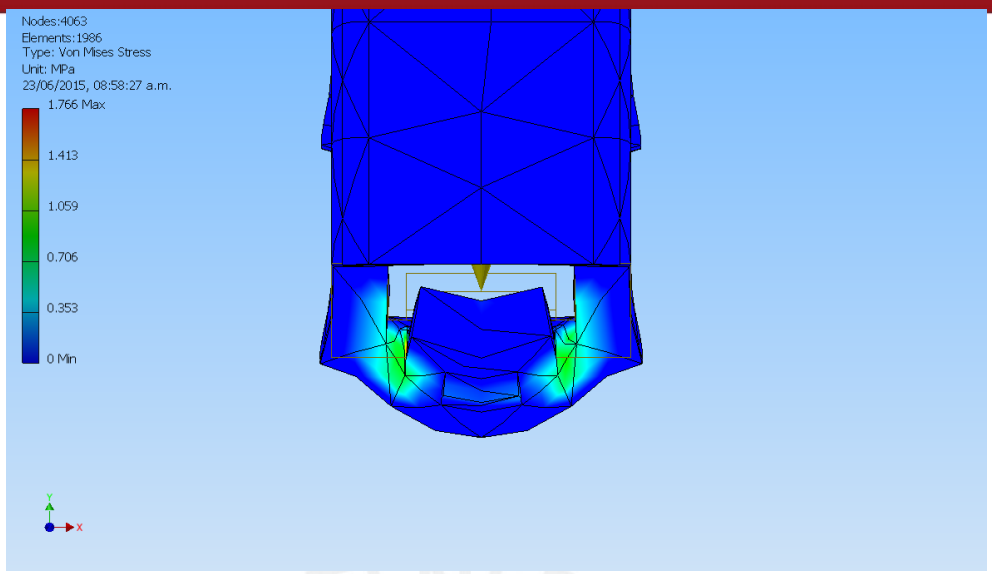


Figura 1.1: Esfuerzos sobre el elemento.

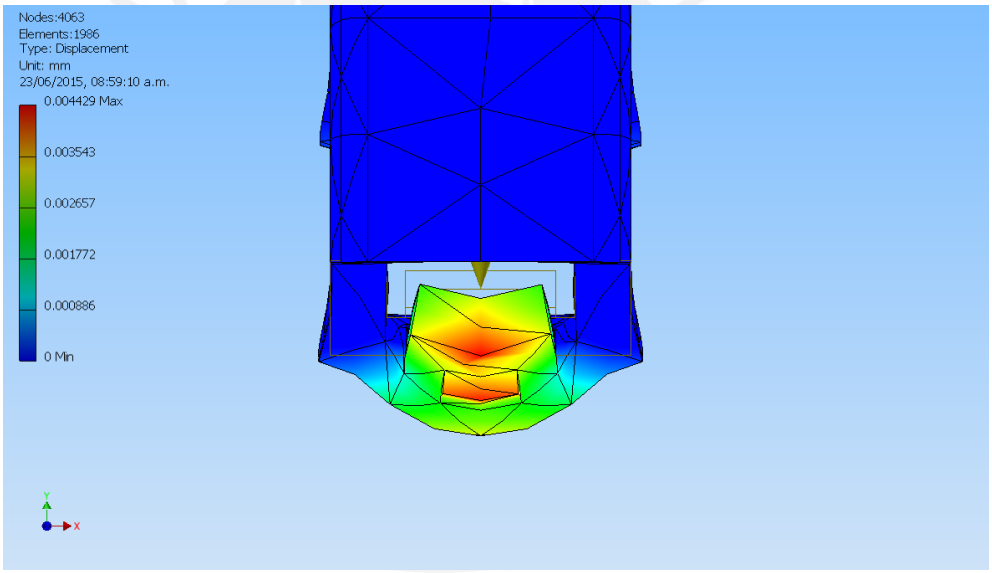


Figura 2.1: Desplazamiento en algunas zonas del elemento.

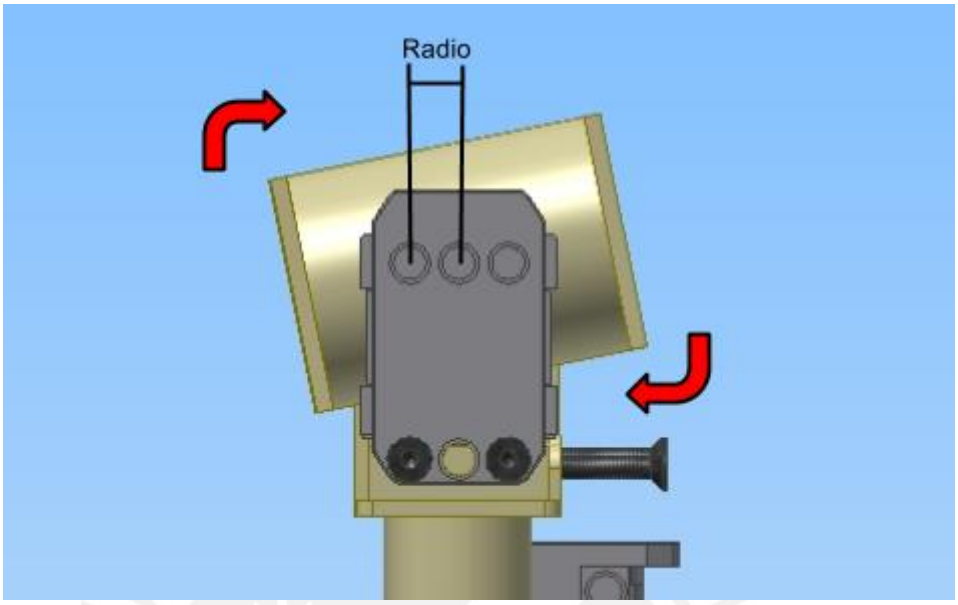
Cálculo de la carga de los motores:

El primer motor debe tener la capacidad de mover el sensor 3D, este tiene un peso de 220 g, su centro de masa pasa aproximadamente por el centro punto de giro del motor y el punto donde el motor se sujeta al sensor 3D está a la distancia al punto central del motor, en pocas palabras, el radio de este.

$$\text{Peso} = 0.220 \text{ kg} * 9.8 \text{ m/s}^2 = 2.156 \text{ N}$$

Radio del motor = 12 mm

Momento para mover el sensor = 0.012 m * 2.156 N = 0.0258 Nm



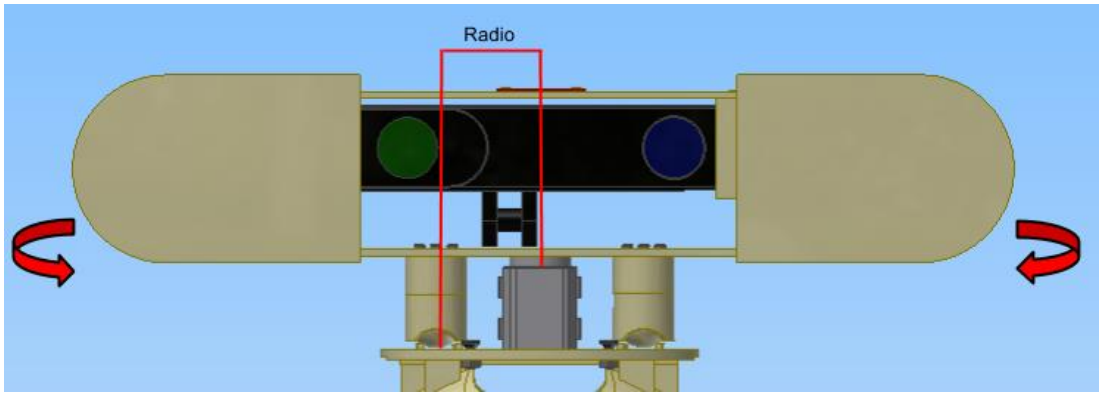
El motor tiene un torque de 0.39 Nm, posee la fuerza suficiente para girar el sensor de lado al lado.

El segundo motor recibe una carga aun mayor ya que recibe el peso del motor, el peso del sensor, otros componentes electrónicos y parte de la estructura. Los apoyos. La carga total que se aplica sobre este motor es de

Peso = 0.323 kg * 9.8 m/s^2 = 3.165 N

Radio del motor = 22 mm

Momento para mover el sensor = 0.022 m * 3.165 N = 0.06963 Nm



Este motor tiene también un torque de 0.39 Nm, posee la fuerza suficiente para girar al sensor y parte de la estructura.

Este motor tiene un alto torque para la tarea que va a realizar, pero se escoge este servo por la alta resolución (0,29°) que tiene, necesaria para que funcione correctamente el sistema de estabilización de imágenes.

Anexo B: Sistemas Electrónicos

Cálculos de corriente:

Las conexiones eléctricas demuestran que la mayoría componentes están conectadas con la tarjeta de procesamiento Odroid la cual trabaja a 5V y 4A, es decir que la corriente máxima de trabajo es de 4A. El consumo de corriente de los componentes conectados no sobrepasa al rango de trabaja de la Odroid, como se demuestra con el siguiente cálculo:

Componentes conectadas a la Odroid	Consumo de corriente
Procesamiento de la Odroid	2000 mA
Xtion Pro	500 mA
Arduino Nano	500 mA
Ventilación	80 mA
Usb Wireless	50 mA
Resultado:	**3130 mA < 4000 mA**

El resultado se suma con el consumo de corriente de cada uno de los servos ya que se conectan directamente a la batería.

Componentes conectados	Consumo de corriente
Odroid	3130 mA
Servo Dynamixel 1	100 mA
Servo Dynamixel 2	100 mA
Resultado:	**3330 mA**

Ahora que tiene el consumo de corriente se puede obtener el tiempo que va a soportar el sistema con la batería ya seleccionada.

$$\text{Tiempo} = \frac{4000\ \text{mAh}}{3330\ \text{mA}}$$

$$\text{Tiempo} = 1.2\ \text{horas}$$

El tiempo resultante es mayor a una hora, la batería del robot dura una hora mientras está trabajando, por tanto el sistema puede funcionar autónomamente según los requerimientos del sistema.

Calculo de cables:

El tamaño de los cables se puede elegir según la corriente máxima que pasa a través de ellos, guiándose de la tabla que se encuentra en el "Handbook of Electronic Tables and Formulas for American Wire Gauge" se puede elegir el número de cable según la norma AWG, norma internacional estándar usada en Perú.

AWG	Diámetro [mm]	Área [mm^2]	Resistencia [Ohms / km]	Corriente Máx. [Amperios]	Frecuencia Máx. [Hz]
12	2.05232	3.31	5.20864	9.3	4150
13	1.8288	2.62	6.56984	7.4	5300
14	1.62814	2.08	8.282	5.9	6700
15	1.45034	1.65	10.44352	4.7	8250
16	1.29032	1.31	13.17248	3.7	11000

*Parte de la tabla de la norma AWG

Según los cálculos de consumo de corriente, se determinó que la corriente máxima es menor a 4 Amperios, por tanto se eligió usar un cable AWG N° 15 ya que soportara el nivel de corriente que consume el sistema.

Cálculo del valor de las resistencias en el divisor de voltaje:

La batería de 7.4 V está conformada por dos baterías de LiPo, cada una de 3.7 V. Una de las baterías, la cual tiene una conexión directa a tierra, se puede medir su voltaje directamente desde uno de los pines ADC del Arduino Nano, esta entrada transforma el voltaje de 0 a 5 V a un valor digital de 0 a 255, si el nivel de voltaje llega a 1 Voltio el led RGB conectado al Arduino cambia de color verde a rojo para indicar que la batería tiene un nivel de energía bajo. La otra batería no puede medirse directamente, por tanto usamos un divisor de voltaje usando dos resistencias, el objetivo es reducir de 7.4 V a 5 V, el ingreso de corriente por este pin es de 40 mA por tanto realizamos los cálculos a continuación:

$$V = I * R_2 \qquad \text{Formula VIR}$$

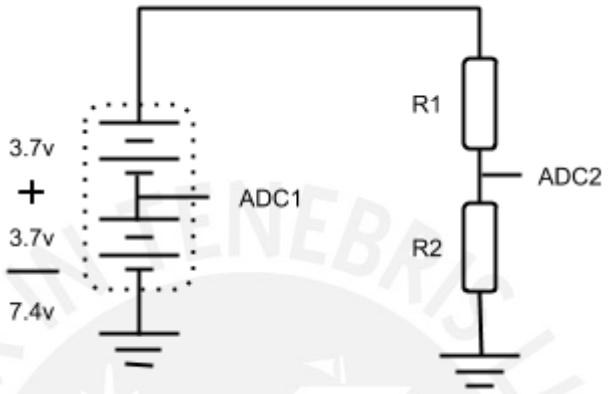
$5 = 40*10^{-3} * R_2$ El voltaje que se requiere es de 5 V

$R_2 = 125\ \Omega$

$V = I * R_1$ Formula VIR

$2.4 = 40*10^{-3} * R_1$ El voltaje que queda 2.4 V

$R_1 = 68\ \Omega$



Divisor de voltaje para la batería

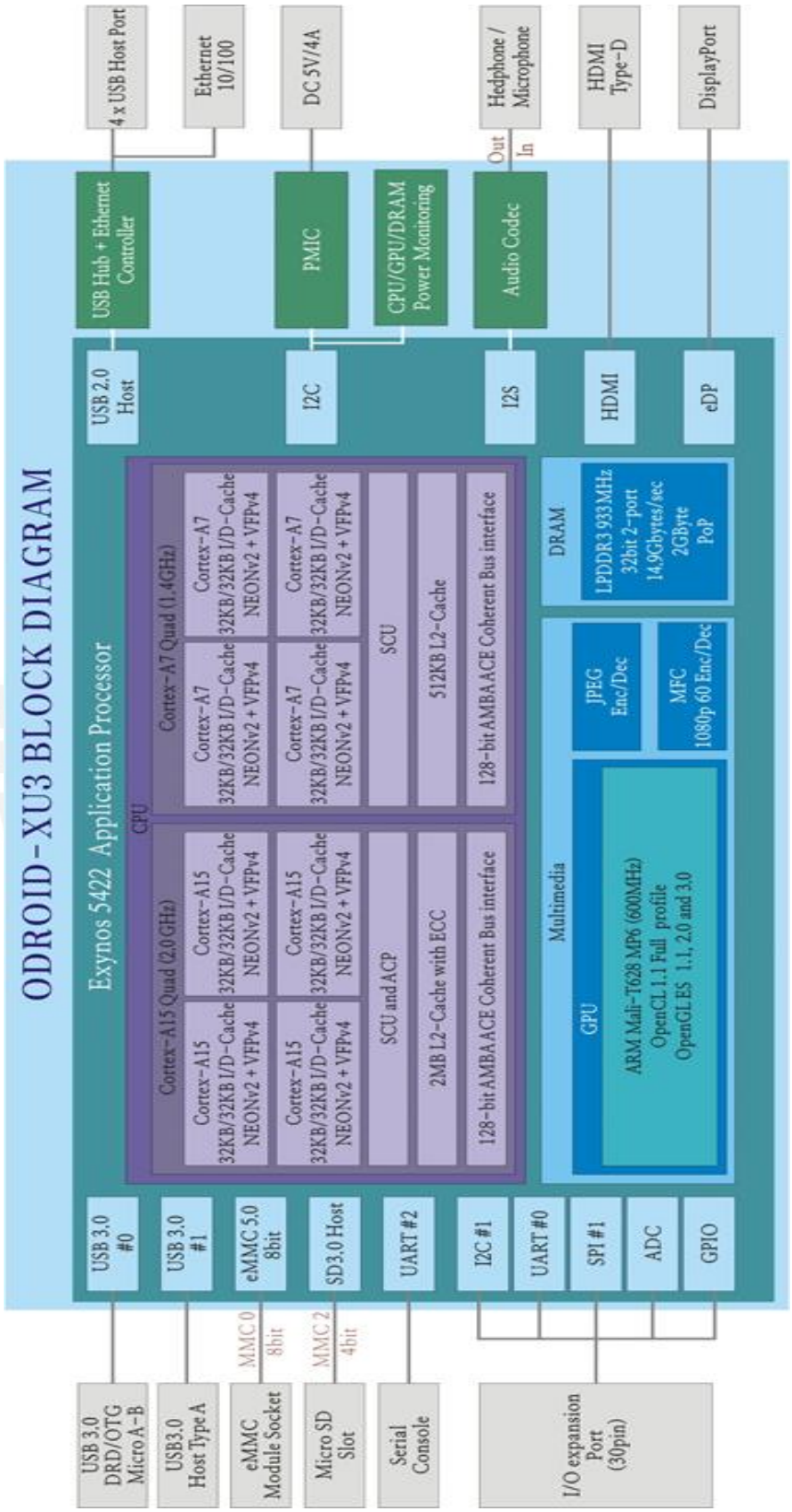
Estos valores no son comerciales, por tanto los ajustamos los valores de las resistencias a dichos valores, el valor más próximo a la R_2 es 120 Ω y para R_1 es 65 Ω, con estos nuevos valores el divisor de voltaje lo divide 7.4 V a 4.8V y 2.6 V. Estos valores son admisibles y ahora el tope de voltaje para el conversor analógico digital será de 4.8 V.

x1	x10	x100	x1.000(K)	x10.000(10K)	x100.000(100K)	x1.000.000(M)
1 Ω	10 Ω	100 Ω	1 KΩ	10 KΩ	100 KΩ	1 M Ω
1,2 Ω	12 Ω	120 Ω	1K2 Ω	12 KΩ	120 KΩ	1M2 Ω
1,5 Ω	15 Ω	150 Ω	1K5 Ω	15 KΩ	150 KΩ	1M5 Ω
1,8 Ω	18 Ω	180 Ω	1K8 Ω	18 KΩ	180 KΩ	1M8 Ω
2,2 Ω	22 Ω	220 Ω	2K2 Ω	22 KΩ	220 KΩ	2M2 Ω
2,7 Ω	27 Ω	270 Ω	2K7 Ω	27 KΩ	270 KΩ	2M7 Ω
3,3 Ω	33 Ω	330 Ω	3K3 Ω	33 KΩ	330 KΩ	3M3 Ω
3,9 Ω	39 Ω	390 Ω	3K9 Ω	39 KΩ	390 KΩ	3M9 Ω
4,7 Ω	47 Ω	470 Ω	4K7 Ω	47 KΩ	470 KΩ	4M7 Ω
5,1 Ω	51 Ω	510 Ω	5K1 Ω	51 KΩ	510 KΩ	5M1 Ω
5,6 Ω	56 Ω	560 Ω	5K6 Ω	56 KΩ	560 KΩ	5M6 Ω
6,8 Ω	68 Ω	680 Ω	6K8 Ω	68 KΩ	680 KΩ	6M8 Ω
8,2 Ω	82 Ω	820 Ω	8K2 Ω	82 KΩ	820 KΩ	8M2 Ω
						10M Ω

Valores comerciales de las resistencias

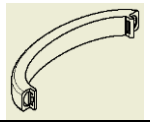
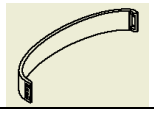
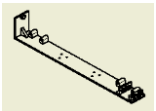
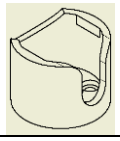
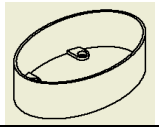
(Fuente: www.sebyc.es)

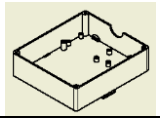
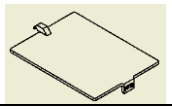
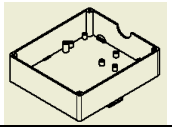
Diagrama de bloques de la computadora embebida
(Fuente http://www.hardkernel.com/)



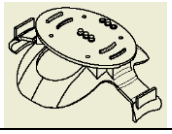
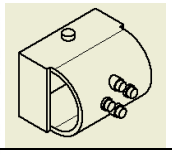
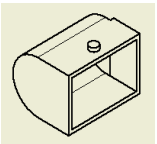
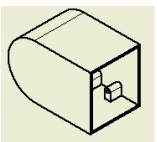
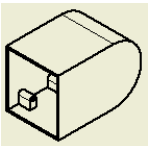
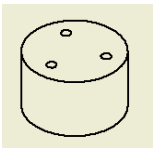
Anexo C: Planos de Piezas y Ensambles

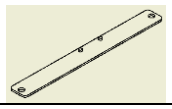
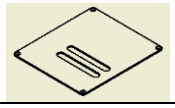
Los planos que se presentan a continuación están organizados por tamaño de lámina, el primer plano A3 es del módulo completo del robot. En los tres siguientes planos siguientes se puede visualizar cada una de las partes del módulo. En la siguiente tabla se muestra la numeración de los planos y la posición en la cual se debe imprimir cada uno, esto tiene una gran influencia en el resultado de las estructuras, la dirección de las hebras no debe ser paralela a las fuerzas que les aplica.

Descripción	Lamina	Posición de impresión
Modulo para Nao	A3 – 1	
Estructura sobre la cabeza	A3 – 2	
Estructura en la espalda	A3 – 3	
Estructura en el pecho	A3 – 4	
Cordón superior	A3 – 5	
Cordón inferior	A3 – 6	
Base del sistema	A3 – 7	
Rueda loca	A3 – 8	
Soporte de billa	A3 – 9	
Carcasa del servo Dynamixel	A3 – 10	

Carcasa de la batería	A3 – 12	
Tapa de la carcasa de la batería	A3 - 13	
Carcasa de la mochila	A3 - 11	

Dirección de las hebras ↘ en las impresiones.

Descripción	Lamina	Posición de impresión(\)
Acople del sistema a la cabeza	A4 – 1	
Sujetador izquierdo	A4 – 2	
Sujetador derecho	A4 – 3	
Carcasa del lado derecho	A4 – 4	
Carcasa del lado izquierdo	A4 – 5	
Cilindro de apoyo	A4 – 6	
Billa	A4 – 7	

Soporte del giroscopio	A4 – 8	
Tapa de la carcasa de la mochila	A4 – 9	

Dirección de las hebras ↘ en las impresiones.

Anexo D: Programas

Código del programa de reconocimiento y reproducción de los movimientos de la persona por el robot en tiempo real:

```
from naoqi import ALProxy    # Importacion de las librerias

pose_to_use = 'Psi'          # Importacion del valor de las posicion de inicio

import cv2
from cv import *             # Importacion de las librerias de opencv
from openni import *         # Importacion de las librerias de openni
import numpy as np           # Importacion de las librerias de matrices en python
import time                  # contador del tiempo
import math                  # Importacion de las librerias matematicas
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt                    # Importacion de las librerias
ploteo
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d

n = 0

fig = plt.figure()                                 # Asignacion de la imagen general
ax = fig.gca(projection='3d')                      # Variable de proyeccion sobre el
ploteo
ax.set_aspect("equal")
zdir = (1, 1, 1)

ax.set_xlim3d(0, 1.5)                              # Limites de los ejes
ax.set_ylim3d(0, 1.5)
ax.set_zlim3d(0, 1.5)
ax.set_xlabel('x')                                 # Nominacion de los ejes de
movimiento
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(-90,-90)
plt.ion()
plt.draw()

class Arrow3D(FancyArrowPatch):
  def __init__(self, xs, ys, zs, *args, **kwargs):
    FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
    self._verts3d = xs, ys, zs

  def draw(self, renderer):
    xs3d, ys3d, zs3d = self._verts3d
    xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
    self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
    FancyArrowPatch.draw(self, renderer)

# Establecer comunicacion y librerias del robot
```

```
naoip = "127.0.0.1"
port = 56174
posture = ALProxy("ALRobotPosture", naoip, port)
motion = ALProxy("ALMotion", naoip, port)

ctx = Context()
ctx.init()

# Creacion del generador de usuarios
user = UserGenerator()
user.create(ctx)

# Obtain the skeleton & pose detection capabilities
skel_cap = user.skeleton_cap
pose_cap = user.pose_detection_cap

# Declaracion de las devoluciones de llamadas para cada usuario y calibracion
def new_user(src, id):
  print "1/4 User {} detected. Looking for pose..." .format(id)
  pose_cap.start_detection(pose_to_use, id)

def pose_detected(src, pose, id):
  print "2/4 Detected pose {} on user {}. Requesting calibration..." .format(pose,id)
  pose_cap.stop_detection(id)
  skel_cap.request_calibration(id, True)

def calibration_start(src, id):
  print "3/4 Calibration started for user {}." .format(id)

def calibration_complete(src, id, status):
  if status == CALIBRATION_STATUS_OK:
    print "4/4 User {} calibrated successfully! Starting to track." .format(id)
    skel_cap.start_tracking(id)
  else:
    print "ERR User {} failed to calibrate. Restarting process." .format(id)
    new_user(user, id)

def lost_user(src, id):
  print "--- User {} lost." .format(id)

# Registrar los usuarios
user.register_user_cb(new_user, lost_user)
pose_cap.register_pose_detected_cb(pose_detected)
skel_cap.register_c_start_cb(calibration_start)
skel_cap.register_c_complete_cb(calibration_complete)

# Establecer el perfil
skel_cap.set_profile(SKEL_PROFILE_ALL)

# Generar el esqueleto
ctx.start_generating_all()
print "0/4 Starting to detect users. Press Ctrl-C to exit."

##############################################################################
```

```
# Funcion que genera las articulaciones
def detect_skeleton():
  global neck, head, rshoul, relbow, rhand, lhand, lelbow, lshoul, torso, lhip, rhip,
lknee, rknee, lfoot, rfoot
  m=0
  neck = np.arange(3)
  head = np.arange(3)
  lshoul = np.arange(3)
  lelbow = np.arange(3)
  lhand = np.arange(3)
  rshoul = np.arange(3)
  relbow = np.arange(3)
  rhand = np.arange(3)
  torso = np.arange(3)
  lhip = np.arange(3)
  rhip = np.arange(3)
  rknee = np.arange(3)
  rfoot = np.arange(3)
  lknee = np.arange(3)
  lfoot = np.arange(3)
# Extraer posición de la cabeza de cada usuario rastreado
  for id in user.users:
    if skel_cap.is_tracking(id):
        m=1
        neck1 = skel_cap.get_joint_position(id, SKEL_NECK)
        head1 = skel_cap.get_joint_position(id, SKEL_HEAD)
        rshoul1=skel_cap.get_joint_position(id, SKEL_RIGHT_SHOULDER)
        relbow1 = skel_cap.get_joint_position(id, SKEL_RIGHT_ELBOW)
        rhand1 = skel_cap.get_joint_position(id, SKEL_RIGHT_HAND)
      lhand1 = skel_cap.get_joint_position(id, SKEL_LEFT_HAND)
        lelbow1 = skel_cap.get_joint_position(id, SKEL_LEFT_ELBOW)
        lshoul1=skel_cap.get_joint_position(id, SKEL_LEFT_SHOULDER)
        torso1 = skel_cap.get_joint_position(id, SKEL_TORSO)
        lhip1 = skel_cap.get_joint_position(id, SKEL_LEFT_HIP)
        rhip1= skel_cap.get_joint_position(id, SKEL_RIGHT_HIP)
        lknee1=skel_cap.get_joint_position(id, SKEL_LEFT_KNEE)
        rknee1=skel_cap.get_joint_position(id, SKEL_RIGHT_KNEE)
        lfoot1=skel_cap.get_joint_position(id, SKEL_LEFT_FOOT)
        rfoot1=skel_cap.get_joint_position(id, SKEL_RIGHT_FOOT)

        neck[0] = (0.000002*(neck1.point[0]/3.5 + 320)**3-
0.0019*(neck1.point[0]/3.5 + 320)**2 + 1.2803*(neck1.point[0]/3.5 + 320) +33.701)
        neck[1] = ((-0.001*(-neck1.point[1]/3.5 + 267.9943)**2+1.1816*(-
neck1.point[1]/3.5 + 267.9943) - 1.072))
        neck[2] = neck1.point[2]/4
        head = (head1.point[0] - neck1.point[0])/4, -(head1.point[1] -
neck1.point[1])/4, -(head1.point[2] - neck1.point[2])/4
        rshoul = (rshoul1.point[0] - neck1.point[0])/4, -(rshoul1.point[1] -
neck1.point[1])/4, -(rshoul1.point[2] - neck1.point[2])/4
        relbow = (relbow1.point[0] - neck1.point[0])/4, -(relbow1.point[1] -
neck1.point[1])/4, -(relbow1.point[2] - neck1.point[2])/4
        rhand = (rhand1.point[0]-neck1.point[0])/4, -(rhand1.point[1]-
neck1.point[1])/4, -(rhand1.point[2]-neck1.point[2])/4
        lhand = (lhand1.point[0]-neck1.point[0])/4, -(lhand1.point[1]-
neck1.point[1])/4, -(lhand1.point[2]-neck1.point[2])/4
```

```
            lelbow = (lelbow1.point[0] - neck1.point[0])/4, -(lelbow1.point[1] -
neck1.point[1])/4, -(lelbow1.point[2] - neck1.point[2])/4
            lshoul = (lshoul1.point[0] - neck1.point[0])/4, -(lshoul1.point[1] -
neck1.point[1])/4, -(lshoul1.point[2] - neck1.point[2])/4
            torso = (torso1.point[0] - neck1.point[0])/4, -(torso1.point[1] -
neck1.point[1])/4, -(torso1.point[2] - neck1.point[2])/4
            lhip = (lhip1.point[0] - neck1.point[0])/4, -(lhip1.point[1] - neck1.point[1])/4, -
(lhip1.point[2] - neck1.point[2])/4
            rhip = (rhip1.point[0] - neck1.point[0])/4, -(rhip1.point[1] - neck1.point[1])/4,
-(rhip1.point[2] - neck1.point[2])/4
            lknee = (lknee1.point[0] - neck1.point[0])/4, -(lknee1.point[1] -
neck1.point[1])/4, -(lknee1.point[2] - neck1.point[2])/4
            rknee = (rknee1.point[0] - neck1.point[0])/4, -(rknee1.point[1] -
neck1.point[1])/4, -(rknee1.point[2] - neck1.point[2])/4
            lfoot = (lfoot1.point[0] - neck1.point[0])/4, -(lfoot1.point[1] -
neck1.point[1])/4, -(lfoot1.point[2] - neck1.point[2])/4
            rfoot = (rfoot1.point[0] - neck1.point[0])/4, -(rfoot1.point[1] -
neck1.point[1])/4, -(rfoot1.point[2] - neck1.point[2])/4

  return m, neck, head, rshoul, relbow, rhand, lhand, lelbow, lshoul, torso, lhip, rhip,
lknee, rknee, lfoot, rfoot

#############################################################################

# Plotear es esqueleto en un plot
def show_skeleton(m, neck1, head1, rshoul1, relbow1, rhand1, lhand1, lelbow1,
lshoul1, torso1, lhip1, rhip1, lknee1, rknee1, lfoot1, rfoot1):
  #Genero la imagen de profundidad
  depth = DepthGenerator()
  depth.create(ctx)
  depth.set_resolution_preset(RES_VGA)
  depth.fps = 30
  newImg = np.fromstring(depth.get_raw_depth_map_8(), "uint8").reshape(480,
640)
  newImg = cv2.cvtColor(newImg,cv2.COLOR_GRAY2BGR)
  #detecto los joints

  if (m == 1):
        cv2.circle(newImg, (int(neck[0]),int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg, (int(head[0]) + int(neck[0]),int(head[1]) +
int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(rshoul[0])+int(neck[0]),int(rshoul[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(relbow[0])+int(neck[0]),int(relbow[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(rhand[0])+int(neck[0]),int(rhand[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(lshoul[0])+int(neck[0]),int(lshoul[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(lelbow[0])+int(neck[0]),int(lelbow[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(lhand[0])+int(neck[0]),int(lhand[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg, (int(torso[0])+int(neck[0]),int(torso[1])+int(neck[1])),5,(255,
255, 255),-1)
```

```
        cv2.circle(newImg, (int(rhip[0])+int(neck[0]),int(rhip[1])+int(neck[1])),5,(255,
255, 255),-1)
        cv2.circle(newImg, (int(lhip[0])+int(neck[0]),int(lhip[1])+int(neck[1])),5,(255,
255, 255),-1)
        cv2.circle(newImg,
(int(rknee[0])+int(neck[0]),int(rknee[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg,
(int(lknee[0])+int(neck[0]),int(lknee[1])+int(neck[1])),5,(255, 255, 255),-1)
        cv2.circle(newImg, (int(rfoot[0])+int(neck[0]),int(rfoot[1])+int(neck[1])),5,(255,
255, 255),-1)
        cv2.circle(newImg, (int(lfoot[0])+int(neck[0]),int(lfoot[1])+int(neck[1])),5,(255,
255, 255),-1)
        cv2.line(newImg,(int(neck[0]),int(neck[1])),(int(head[0]) +
int(neck[0]),int(head[1]) + int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(neck[0]),int(neck[1])),(int(lshoul[0])+int(neck[0]),int(lshou
l[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(neck[0]),int(neck[1])),(int(rshoul[0])+int(neck[0]),int(rsho
ul[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(rshoul[0])+int(neck[0]),int(rshoul[1])+int(neck[1])),(int(tor
so[0])+int(neck[0]),int(torso[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(rshoul[0])+int(neck[0]),int(rshoul[1])+int(neck[1])),(int(rel
bow[0])+int(neck[0]),int(relbow[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(relbow[0])+int(neck[0]),int(relbow[1])+int(neck[1])),(int(rh
and[0])+int(neck[0]),int(rhand[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(lshoul[0])+int(neck[0]),int(lshoul[1])+int(neck[1])),(int(lelb
ow[0])+int(neck[0]),int(lelbow[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(lelbow[0])+int(neck[0]),int(lelbow[1])+int(neck[1])),(int(lh
and[0])+int(neck[0]),int(lhand[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(lshoul[0])+int(neck[0]),int(lshoul[1])+int(neck[1])),(int(tors
o[0])+int(neck[0]),int(torso[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(torso[0])+int(neck[0]),int(torso[1])+int(neck[1])),(int(rhip[
0])+int(neck[0]),int(rhip[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(torso[0])+int(neck[0]),int(torso[1])+int(neck[1])),(int(lhip[0
])+int(neck[0]),int(lhip[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(rhip[0])+int(neck[0]),int(rhip[1])+int(neck[1])),(int(rknee[0
])+int(neck[0]),int(rknee[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(lhip[0])+int(neck[0]),int(lhip[1])+int(neck[1])),(int(lknee[0]
)+int(neck[0]),int(lknee[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(rknee[0])+int(neck[0]),int(rknee[1])+int(neck[1])),(int(rfoo
t[0])+int(neck[0]),int(rfoot[1])+int(neck[1])),(0,255,0),2)
```

```
        cv2.line(newImg,(int(lknee[0])+int(neck[0]),int(lknee[1])+int(neck[1])),(int(lfoot
[0])+int(neck[0]),int(lfoot[1])+int(neck[1])),(0,255,0),2)

        cv2.line(newImg,(int(lhip[0])+int(neck[0]),int(lhip[1])+int(neck[1])),(int(rhip[0])+
int(neck[0]),int(rhip[1])+int(neck[1])),(0,255,0),2)

   newImg = cv2.flip (newImg, 1)
   cv2.imshow("Main", newImg)

#########################################################################

# Ecuaciones para las transformaciones matematicas del brazo izquierdo
def equations_lhand_x(xp,anglelbowxz):
   alpha = math.atan2(xp[1],xp[0])
   beta = math.atan2(xp[2],math.sqrt(xp[0]**2+xp[1]**2))
   yp = [-math.sin(beta)*math.cos(alpha), -math.sin(beta)*math.sin(alpha),
math.cos(beta)]
   zp = [xp[1]*yp[2]-xp[2]*yp[1], -xp[0]*yp[2]+xp[2]*yp[0], xp[0]*yp[1]-xp[1]*yp[0]]
   theta = 180*math.pi/180
   yp_r_x = yp[0]*(math.cos(theta-anglelbowxz)+(xp[0]**2)*(1-math.cos(theta-
anglelbowxz)))+yp[1]*(xp[0]*xp[1]*(1-math.cos(theta-anglelbowxz))-
xp[2]*math.sin(theta-anglelbowxz))+yp[2]*(xp[0]*xp[2]*(1-math.cos(theta-
anglelbowxz))-xp[1]*math.sin(theta-anglelbowxz))
   yp_r_y = yp[0]*(xp[1]*xp[0]*(1-math.cos(theta-anglelbowxz))-xp[2]*math.sin(theta-
anglelbowxz))+yp[1]*(math.cos(theta-anglelbowxz)+(xp[1]**2)*(1-math.cos(theta-
anglelbowxz)))+yp[2]*(xp[2]*xp[1]*(1-math.cos(theta-anglelbowxz))-
xp[0]*math.sin(theta-anglelbowxz))
   yp_r_z = yp[0]*(xp[2]*xp[0]*(1-math.cos(theta-anglelbowxz))-xp[1]*math.sin(theta-
anglelbowxz))+yp[1]*(xp[2]*xp[1]*(1-math.cos(theta-anglelbowxz))-
xp[0]*math.sin(theta-anglelbowxz))+yp[2]*(math.cos(theta-
anglelbowxz)+(xp[2]**2)*(1-math.cos(theta-anglelbowxz)))
   yp_r = [yp_r_x,yp_r_y,yp_r_z]
   zp_r = [xp[1]*yp_r[2]-xp[2]*yp_r[1], -xp[0]*yp_r[2]+xp[2]*yp_r[0], xp[0]*yp_r[1]-
xp[1]*yp_r[0]]
   return yp_r,zp_r

#########################################################################

# Ecuaciones para las transformaciones matematicas del brazo derecho
def equations_rhand_x(xp,angrelbowxz):
   beta = math.atan2(xp[2],math.sqrt(xp[0]**2+xp[1]**2))
   yp = [-math.sin(beta)*math.cos(alpha), -math.sin(beta)*math.sin(alpha),
math.cos(beta)]
   zp = [xp[1]*yp[2]-xp[2]*yp[1], -xp[0]*yp[2]+xp[2]*yp[0], xp[0]*yp[1]-xp[1]*yp[0]]
   theta = 180*math.pi/180
   yp_r_x = yp[0]*(math.cos(theta-angrelbowxz)+(xp[0]**2)*(1-math.cos(theta-
angrelbowxz)))+yp[1]*(xp[0]*xp[1]*(1-math.cos(theta-angrelbowxz))-
xp[2]*math.sin(theta-angrelbowxz))+yp[2]*(xp[0]*xp[2]*(1-math.cos(theta-
angrelbowxz))-xp[1]*math.sin(theta-angrelbowxz))
   yp_r_y = yp[0]*(xp[1]*xp[0]*(1-math.cos(theta-angrelbowxz))-xp[2]*math.sin(theta-
angrelbowxz))+yp[1]*(math.cos(theta-angrelbowxz)+(xp[1]**2)*(1-math.cos(theta-
angrelbowxz)))+yp[2]*(xp[2]*xp[1]*(1-math.cos(theta-angrelbowxz))-
xp[0]*math.sin(theta-angrelbowxz))
```

```
    yp_r_z = yp[0]*(xp[2]*xp[0]*(1-math.cos(theta-angrelbowxz))-xp[1]*math.sin(theta-angrelbowxz))+yp[1]*(xp[2]*xp[1]*(1-math.cos(theta-angrelbowxz))-xp[0]*math.sin(theta-angrelbowxz))+yp[2]*(math.cos(theta-angrelbowxz)+(xp[2]**2)*(1-math.cos(theta-angrelbowxz)))
    yp_r = [yp_r_x,yp_r_y,yp_r_z]
    zp_r = [xp[1]*yp_r[2]-xp[2]*yp_r[1], -xp[0]*yp_r[2]+xp[2]*yp_r[0], xp[0]*yp_r[1]-xp[1]*yp_r[0]]
    return yp_r,zp_r

#########################################################################

# Representacion de las coordenadas en una matriz 2D
def show_3d_axis(xp,yp,zp):
    ax.text(xp[0]+0.75, xp[1]+0.75, xp[2]+0.75, 'xp', zdir)
    ax.text(yp[0]+0.75, yp[1]+0.75, yp[2]+0.75,'yp', zdir)
    ax.text(zp[0]+0.75, zp[1]+0.75, zp[2]+0.75, 'zp', zdir)
    ax.text(1 + 0.75, 0.75, 0.75, 'x', zdir)
    ax.text(0.75, 1+0.75, 0.75,'y', zdir)
    ax.text(0.75, 0.75, 1+0.75, 'z', zdir)

    a = Arrow3D([0.75,xp[0]+0.75],[0.75, xp[1]+0.75],[0.75,xp[2]+0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")
    b = Arrow3D([0.75,yp[0]+0.75],[0.75,yp[1]+0.75],[0.75,yp[2]+0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")
    c = Arrow3D([0.75,zp[0]+0.75],[0.75,zp[1]+0.75],[0.75,zp[2]+0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")
    d = Arrow3D([0.75,1+0.75],[0.75, 0.75],[0.75,0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")
    e = Arrow3D([0.75,0.75],[0.75, 1+0.75],[0.75,0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")
    f = Arrow3D([0.75,0.75],[0.75, 0.75],[0.75,1+0.75], mutation_scale=30, lw=1, arrowstyle="-|>", color="k")

    ax.add_artist(a)
    ax.add_artist(b)
    ax.add_artist(c)
    ax.add_artist(d)
    ax.add_artist(e)
    ax.add_artist(f)

    plt.draw()
    plt.pause(0.0000000000001)
    plt.cla()

#########################################################################

# Transformaciones matematicas para el brazo izquierdo
def Operations_larm(lshoul,lelbow,lhand):
    vect_lhandlelbow = [lhand[0]-lelbow[0], lhand[1]-lelbow[1], lhand[2]-lelbow[2]]
    abs_vect_lhandlelbow = math.sqrt(vect_lhandlelbow[0]**2+vect_lhandlelbow[1]**2+vect_lhandlelbow[2]**2)
    vect_lhandlelbow = [vect_lhandlelbow[0]/abs_vect_lhandlelbow,vect_lhandlelbow[1]/abs_vect_lhandlelbow,vect_lhandlelbow[2]/abs_vect_lhandlelbow]
```

```
    abs_vect_lelbow = math.sqrt((lelbow[0]-lshoul[0])**2+(lelbow[1]-
lshoul[1])**2+((lelbow[2]-lshoul[2])**2))
    anglelbowxy = -math.asin(abs((lelbow[2]-lshoul[2]))/abs_vect_lelbow)
    anglelbowyz = -math.asin(abs((lelbow[0]-lshoul[0]))/abs_vect_lelbow)
    anglelbowxz = math.asin(abs((lelbow[1]-lshoul[1]))/abs_vect_lelbow)
    if  (lelbow[2] >= lshoul[2]):
        anglelbowxy = math.asin(abs((lelbow[2]-lshoul[2]))/abs_vect_lelbow)

    if  (lelbow[0] >= lshoul[0]):
        anglelbowyz = math.asin(abs((lelbow[0]-lshoul[0]))/abs_vect_lelbow)

    if (lelbow[1] <= lshoul[1]):
        anglelbowxz = -math.asin(abs((lelbow[1]-lshoul[1]))/abs_vect_lelbow)
    xp = [(lelbow[0]-lshoul[0])/abs_vect_lelbow,(lelbow[1]-lshoul[1])/abs_vect_lelbow,
(lelbow[2]-lshoul[2])/abs_vect_lelbow]
    yp_r,zp_r = equations_lhand_x(xp,anglelbowxz)
    if vect_lhandlelbow[1] > yp_r[1]:
        anglhand_xpyp = -
abs(zp_r[0]*vect_lhandlelbow[0]+zp_r[1]*vect_lhandlelbow[1]+zp_r[2]*vect_lhandlel
bow[2])
    else:
            anglhand_xpyp =
abs(zp_r[0]*vect_lhandlelbow[0]+zp_r[1]*vect_lhandlelbow[1]+zp_r[2]*vect_lhandlel
bow[2])
    f1 = vect_lhandlelbow[1]*zp_r[2]-vect_lhandlelbow[2]*zp_r[1]
    f2 = vect_lhandlelbow[0]*zp_r[2]-vect_lhandlelbow[2]*zp_r[0]
    f3 = vect_lhandlelbow[0]*zp_r[1]-vect_lhandlelbow[1]*zp_r[0]
    proy_xpyp = [(-(zp_r[1]*f3+zp_r[2]*f2)),(zp_r[0]*f3-
zp_r[2]*f1),(zp_r[0]*(zp_r[1]*f3+zp_r[2]*f2)-zp_r[1]*(zp_r[0]*f3-zp_r[2]*f1))/zp_r[2]]
    abs_proy_xpyp = math.sqrt(proy_xpyp[0]**2+proy_xpyp[1]**2+proy_xpyp[2]**2)
    proy_xpyp = [proy_xpyp[0]/abs_proy_xpyp,proy_xpyp[1]/abs_proy_xpyp
,proy_xpyp[2]/abs_proy_xpyp ]
    anglhand_xp =
math.acos(abs(xp[0]*proy_xpyp[0]+xp[1]*proy_xpyp[1]+xp[2]*proy_xpyp[2]))
    return xp, yp_r, zp_r, anglelbowxy, anglelbowyz, anglelbowxz, anglhand_xpyp,
anglhand_xp

#########################################################################

# Transformaciones matematicas para el brazo derecho para el brazo
def Operations_rarm(rshoul,relbow,rhand):
    vect_rhandrelbow = [rhand[0]-relbow[0], rhand[1]-relbow[1], rhand[2]-relbow[2]]
    abs_vect_rhandrelbow =
math.sqrt(vect_rhandrelbow[0]**2+vect_rhandrelbow[1]**2+vect_rhandrelbow[2]**2)
    vect_rhandrelbow =
[vect_rhandrelbow[0]/abs_vect_rhandrelbow,vect_rhandrelbow[1]/abs_vect_rhandre
lbow,vect_rhandrelbow[2]/abs_vect_rhandrelbow]
    abs_vect_relbow = math.sqrt((relbow[0]-rshoul[0])**2+(relbow[1]-
rshoul[1])**2+((relbow[2]-rshoul[2])**2))
    angrelbowxy = -math.asin(abs((relbow[2]-rshoul[2]))/abs_vect_relbow)
    angrelbowyz = -math.asin(abs((relbow[0]-rshoul[0]))/abs_vect_relbow)
    angrelbowxz = math.asin(abs((relbow[1]-rshoul[1]))/abs_vect_relbow)
    if  (relbow[2] >= rshoul[2]):
        angrelbowxy = math.asin(abs((relbow[2]-rshoul[2]))/abs_vect_relbow)
```

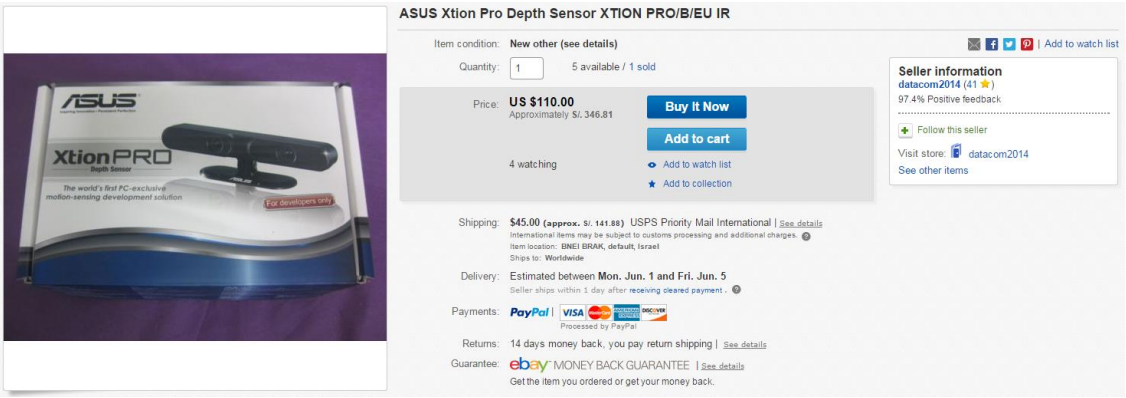
```
  if  (relbow[0] >= rshoul[0]):
      angrelbowyz = math.asin(abs((relbow[0]-rshoul[0]))/abs_vect_relbow)

  if (relbow[1] <= rshoul[1]):
      angrelbowxz = -math.asin(abs((relbow[1]-rshoul[1]))/abs_vect_relbow)
  xp = [(relbow[0]-rshoul[0])/abs_vect_relbow,(relbow[1]-rshoul[1])/abs_vect_relbow,
(relbow[2]-rshoul[2])/abs_vect_relbow]
  yp_r,zp_r = equations_rhand_x(xp,angrelbowxz)
  if vect_rhandrelbow[1] > yp_r[1]:
      angrhand_xpyp = -
abs(zp_r[0]*vect_rhandrelbow[0]+zp_r[1]*vect_rhandrelbow[1]+zp_r[2]*vect_rhandre
lbow[2])
  else:
          angrhand_xpyp =
abs(zp_r[0]*vect_rhandrelbow[0]+zp_r[1]*vect_rhandrelbow[1]+zp_r[2]*vect_rhandre
lbow[2])
  f1 = vect_rhandrelbow[1]*zp_r[2]-vect_rhandrelbow[2]*zp_r[1]
  f2 = vect_rhandrelbow[0]*zp_r[2]-vect_rhandrelbow[2]*zp_r[0]
  f3 = vect_rhandrelbow[0]*zp_r[1]-vect_rhandrelbow[1]*zp_r[0]
  proy_xpyp = [(-(zp_r[1]*f3+zp_r[2]*f2)),(zp_r[0]*f3-
zp_r[2]*f1),(zp_r[0]*(zp_r[1]*f3+zp_r[2]*f2)-zp_r[1]*(zp_r[0]*f3-zp_r[2]*f1))/zp_r[2]]
  abs_proy_xpyp = math.sqrt(proy_xpyp[0]**2+proy_xpyp[1]**2+proy_xpyp[2]**2)
  proy_xpyp = [proy_xpyp[0]/abs_proy_xpyp,proy_xpyp[1]/abs_proy_xpyp
,proy_xpyp[2]/abs_proy_xpyp ]
  angrhand_xp =
math.acos(abs(xp[0]*proy_xpyp[0]+xp[1]*proy_xpyp[1]+xp[2]*proy_xpyp[2]))
  return xp, yp_r, zp_r, angrelbowxy, angrelbowyz, angrelbowxz, angrhand_xpyp,
angrhand_xp

#########################################################################

# Asignacion de las variables al robot
def NAO(m, neck,lshoul,lelbow,lhand):
  if (m == 1):
      names = list()
      times = list()
      keys = list()

          l_xp, l_yp_r, l_zp_r, anglelbowxy, anglelbowyz, anglelbowxz,
anglhand_xpyp, anglhand_xp = Operations_larm(lshoul,lelbow,lhand)
          r_xp, r_yp_r, r_zp_r, angrelbowxy, angrelbowyz, angrelbowxz,
angrhand_xpyp, angrhand_xp = Operations_rarm(rshoul,relbow,rhand)

      names.append("LElbowRoll")
      times.append([0.72])
      keys.append([-angrhand_xp])
      names.append("LElbowYaw")
      times.append([0.72])
      keys.append([-angrhand_xpyp])
      names.append("LHand")
      times.append([0.72])
      keys.append([1])
      names.append("LShoulderPitch")
      times.append([0.72])
      keys.append([angrelbowxz])
```

```
        names.append("LShoulderRoll")
        times.append([0.72])
        keys.append([0.84351922587*angrelbowyz])
        names.append("LWristYaw")
        times.append([0.72])
        keys.append([-1.82346])
        names.append("RShoulderPitch")
        times.append([0.72])
        keys.append([anglelbowxz])
        names.append("RElbowYaw")
        times.append([0.72])
        keys.append([anglhand_xpyp])
        names.append("RHand")
        times.append([0.72])
        keys.append([1])
        names.append("RElbowRoll")
        times.append([0.72])
        keys.append([anglhand_xp])
        names.append("RShoulderRoll")
        times.append([0.72])
        keys.append([0.84351922587*anglelbowyz])
        names.append("RWristYaw")
        times.append([0.72])
        keys.append([1.82346])

        motion.angleInterpolation(names, keys, times, True)

##########################################################################

#Programa principal

while True:
  # Update to next frame
  ctx.wait_and_update_all()

  m, fneck, fhead, frshoul, frelbow, frhand, flhand, flelbow, flshoul, ftorso, flhip, frhip,
flknee, frknee, flfoot, frfoot = detect_skeleton()
  show_skeleton(m, fneck, fhead, frshoul, frelbow, frhand, flhand, flelbow, flshoul,
ftorso, flhip, frhip, flknee, frknee, flfoot, frfoot)
  if n ==5:
        NAO(m, fneck,flshoul,flelbow,flhand)
        n = 0
  n = n + 1
  k = cv2.waitKey(1) & 0xFF
  if k == 27:
        break
```

Anexo E: Proformas

Batería:

Sensor 3D:

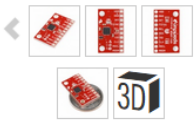


Servo Dynamixel:

Computadora embebida:

Acelerómetro/Giroscopio:

HOME / PRODUCT CATEGORIES / IMU / SPARKFUN TRIPLE AXIS ACCELEROMETER AND GYRO BREAKOUT - MPU-6050



images are CC BY-NC-SA 3.0

3D Download: Sketchup, STL, Blender

SparkFun Triple Axis Accelerometer and Gyro Breakout - MPU-6050

SEN-11028 ROHS

Description: The MPU-6050 is a serious little piece of motion processing tech! By combining a MEMS 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor™ (DMP™) capable of processing complex 9-axis MotionFusion algorithms, the MPU-6050 does away with the cross-axis alignment problems that can creep up on discrete parts.

Our breakout board for the MPU-6050 makes this tiny QFN package easy to work into your project. Every pin you need to get up and running is broken out to 0.1" headers, including the auxiliary master I2C bus which allows the MPU-6050 to access external magnetometers and other sensors.

Having a hard time picking an IMU? Our Accelerometer, Gyro, and IMU Buying Guide might help!

Dimensions: 1 x 0.6 x 0.09" (25.5 x 15.2 x 2.48mm)

Features:

- I2C Digital-output of 6 or 9-axis MotionFusion data in rotation

$39.95

ADD TO CART

1	quantity
	248 in stock
$39.95	1+ units
$37.95	10+ units
$35.96	25+ units
$33.96	100+ units

SHARE

FAVORITE 1

WISH LIST

Arduino Nano:

OpenHardware.pe | ARDUINO ORIGINAL Distribuidor oficial en Perú | SENSORES GROVE Distribuidor oficial en Perú | ENVÍO EN 24H A todo Perú con Olva Courrier

CATEGORIES | Search entire store here... | All Categories | 0 ARTICULO(S) - $0.00

INICIO > ARDUINO > ARDUINO BOARDS > ARDUINO NANO

CATEGORÍAS

- Arduino (60)
  - Arduino Boards (19)
  - Arduino Shields (10)
- Arduino Kits (8)
- Shields (70)
- Grove (80)
- Raspberry PI (13)
- Sensores (172)
- Componentes (59)
- Robótica (164)
- Impresión 3D (38)
- Maker Zone (4)

Hover | Large

ARDUINO NANO

- Marca: Arduino
- Código Producto: A000005
- Disponibilidad: 3

PRECIO: $54.28

Sin Impuesto: $46.00

Cantidad: 1

COMPRAR + COMPARAR + LISTA DE REGALOS

Impresión 3D:

VEO IMPRESIÓN 3D

PUCP

Lima 16 de Junio de 2015

SOLICITUD DE COTIZACIÓN

Estimado

Nombre: José Alexander López Manrique
Codigo: 20092130

Ref. Impresión 3d de prototipo en plástico
Lima.-

NOMBRE	SERVICIO	PRECIO	MATERIAL
José Alexander López Manrique	Sistema para robot NAO	S/. 500	ABS Fortus 400mc / ABS Da Vinci

Montos expresados en nuevos soles (PEN), incluye IGV.

Forma de pago
- Adelantado
- Tesorería de la Pontificia Universidad Católica del Perú

Tiempo de entrega
3 días utiles

Atentamente,

Jennifer Wong Poggi
Responsable del área de Impresión 3D
Pontificia Universidad Católica del Perú

Pontificia Universidad Católica del Perú | Av. Universitaria 1801, San Miguel, Lima 32, Perú | Teléfono (511) 626-2000 Anexo 3945 |
Correo: veo3d@pucp.pe