Apéndices

Apéndice A

Plano QuadRotor

A continuación se presentan, en un mismo plano, los elementos del quadrotor junto con sus dimensiones principales.

La acotación de los elementos no ha pretendido ser completa, al no ser objeto del proyecto el diseño del quadrotor, de modo que solo se presentan aquellas dimensiones que se consideran de mayor importancia para futuras aplicaciones o comprobaciones que se realicen con dicho modelo y/o su simulador.

Apéndice B

Códigos de programación - QuadRotor

B.1 quadrotor.urdf

```
<?xml version="1.0"?>

<!-- Versión QuadRotor helices fijas + Soportes + camara + plugins
de Gazebo -->

<!-- Se anaden los archivos de cabecera para los plugins--!>

<robot
    xmlns:controller="http://playerstage.sourceforge.net/gazebo/xmlschema
        /#controller"
    xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#
        sensor"
    xmlns:interface="http://playerstage.sourceforge.net/gazebo/xmlschema
        /#interface"
    name="quadrotor">

<!--**********************BASE**********************-->
<link name="Base">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_base_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_base_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 5.9e-3" rpy="0 0 0"/>
    <mass value="1.578"/>
    <inertia ixx="0.01632655" ixy="0.063e-7" ixz="0.124e-7"
            iyy="0.01632655" iyz="0.113e-7" izz="0.02642555"/>
```

```
  </inertial>
</link>

<!--******************MOTORES******************-->
<link name="motor_1">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.020" rpy="0 0 0"/>
    <mass value="0.033"/>
    <inertia ixx="63.23e-7" ixy="0.0" ixz="0.0"
             iyy="63.23e-7" iyz="0.0" izz="63.23e-7"/>
  </inertial>
</link>
<joint name="base_a_motor_1" type="fixed">
  <parent link="Base"/>
  <child link="motor_1"/>
  <origin rpy="0 0 0" xyz="0.2873 0 0.015"/>
</joint>

<link name="motor_4">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.020" rpy="0 0 0"/>
    <mass value="0.033"/>
    <inertia ixx="63.23e-7" ixy="0.0" ixz="0.0"
             iyy="63.23e-7" iyz="0.0" izz="63.23e-7"/>
  </inertial>
</link>
<joint name="base_a_motor_4" type="fixed">
  <parent link="Base"/>
  <child link="motor_4"/>
```

```
  <origin rpy="0 0 1.57079" xyz="0 0.2873 0.015"/>
</joint>

<link name="motor_3">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.020" rpy="0 0 0"/>
    <mass value="0.033"/>
    <inertia ixx="63.23e-7" ixy="0.0" ixz="0.0"
             iyy="63.23e-7" iyz="0.0" izz="63.23e-7"/>
  </inertial>
</link>
<joint name="base_a_motor_3" type="fixed">
  <parent link="Base"/>
  <child link="motor_3"/>
  <origin rpy="0 0 3.14159" xyz="-0.2873 0 0.015"/>
</joint>

<link name="motor_2">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/pelican_motor_escala.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.020" rpy="0 0 0"/>
    <mass value="0.033"/>
    <inertia ixx="63.23e-7" ixy="0.0" ixz="0.0"
             iyy="63.23e-7" iyz="0.0" izz="63.23e-7"/>
  </inertial>
</link>
<joint name="base_a_motor_2" type="fixed">
  <parent link="Base"/>
  <child link="motor_2"/>
  <origin rpy="0 0 4.71239" xyz="0 -0.2873 0.015"/>
```

```
</joint>

<!--******************HELICES******************-->
<link name="helice_1">
  <visual>
    <geometry>
     <mesh filename="package://modelquad/visual/
         pelican_helice_escala_amarillo.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
     <mesh filename="package://modelquad/visual/
         pelican_helice_escala_amarillo.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.022" rpy="0 0 0"/>
    <mass value="0.005"/>
    <inertia ixx="2.6e-7" ixy="4.13e-7" ixz="0.0"
             iyy="199.012e-7" iyz="0.0" izz="198.388e-7"/>
  </inertial>
</link>
<joint name="motor_a_helice_1" type="fixed">
  <parent link="motor_1"/>
  <child link="helice_1"/>
  <origin rpy="0 0 0" xyz="0 0 0.04"/>
</joint>

<link name="helice_4">
  <visual>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
   <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.022" rpy="0 0 0"/>
    <mass value="0.005"/>
    <inertia ixx="2.6e-7" ixy="4.13e-7" ixz="0.0"
             iyy="199.012e-7" iyz="0.0" izz="198.388e-7"/>
  </inertial>
</link>
<joint name="motor_a_helice_4" type="fixed">
  <parent link="motor_4"/>
```

```
  <child link="helice_4"/>
  <origin rpy="0 0 0" xyz="0 0 0.04"/>
</joint>

<link name="helice_3">
  <visual>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
   <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.022" rpy="0 0 0"/>
    <mass value="0.005"/>
    <inertia ixx="2.6e-7" ixy="4.13e-7" ixz="0.0"
             iyy="199.012e-7" iyz="0.0" izz="198.388e-7"/>
  </inertial>
</link>
<joint name="motor_a_helice_3" type="fixed">
  <parent link="motor_3"/>
  <child link="helice_3"/>
  <origin rpy="0 0 0" xyz="0 0 0.04"/>
</joint>

<link name="helice_2">
  <visual>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
   <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
   <geometry>
    <mesh filename="package://modelquad/visual/pelican_helice_escala.dae"
          scale="0.0255 0.0255 0.0255"/>
   </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0.022" rpy="0 0 0"/>
    <mass value="0.005"/>
    <inertia ixx="2.6e-7" ixy="4.13e-7" ixz="0.0"
             iyy="199.012e-7" iyz="0.0" izz="198.388e-7"/>
  </inertial>
</link>
<joint name="motor_a_helice_2" type="fixed">
  <parent link="motor_2"/>
  <child link="helice_2"/>
```

```
  <origin rpy="0 0 0" xyz="0 0 0.04"/>
</joint>

<!--******************SOPORTES******************-->
<link name="Soporte_1">
  <visual>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
    <material name="blue">
     <color rgba="0 0.5 0.5 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0"
             iyy="0.0" iyz="0.0" izz="0.0"/>
  </inertial>
</link>
<joint name="base_a_Soporte_1" type="fixed" >
  <parent link="Base" />
  <child link="Soporte_1" />
  <origin rpy="0 0 0" xyz="0.2873 0 -0.242"/>
</joint>
<gazebo reference="Soporte_1">
  <material>Gazebo/Blue</material>
</gazebo>

<link name="Soporte_2">
  <visual>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
    <material name="blue">
     <color rgba="0 0.5 0.5 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0"
             iyy="0.0" iyz="0.0" izz="0.0"/>
  </inertial>
</link>
```

```
<joint name="base_a_Soporte_2" type="fixed" >
  <parent link="Base" />
  <child link="Soporte_2" />
  <origin rpy="0 0 0" xyz="0 0.2873 -0.242"/>
</joint>
<gazebo reference="Soporte_2">
  <material>Gazebo/Blue</material>
</gazebo>

<link name="Soporte_3">
  <visual>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
    <material name="blue">
     <color rgba="0 0.5 0.5 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0"
             iyy="0.0" iyz="0.0" izz="0.0"/>
  </inertial>
</link>
<joint name="base_a_Soporte_3" type="fixed" >
  <parent link="Base" />
  <child link="Soporte_3" />
  <origin rpy="0 0 0" xyz="-0.2873 0 -0.242"/>
</joint>
<gazebo reference="Soporte_3">
  <material>Gazebo/Blue</material>
</gazebo>

<link name="Soporte_4">
  <visual>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
    <material name="blue">
     <color rgba="0 0.5 0.5 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
     <cylinder length="0.5" radius="0.005"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
```

```
    <mass value="0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0"
             iyy="0.0" iyz="0.0" izz="0.0"/>
  </inertial>
</link>
<joint name="base_a_Soporte_4" type="fixed" >
  <parent link="Base" />
  <child link="Soporte_4" />
  <origin rpy="0 0 0" xyz="0 -0.2873 -0.242"/>
</joint>
<gazebo reference="Soporte_4">
  <material>Gazebo/Blue</material>
</gazebo>

<!--******************CAMARA******************-->
<link name="Camara">
 <inertial>
   <mass value="1e-100" />
   <origin xyz="0 0 0" />
   <inertia ixx="1e-100" ixy="0" ixz="0"
            iyy="1e-100" iyz="0" izz="1e-100" />
 </inertial>
 <visual>
   <origin xyz="0 0 -0.025" rpy="0 0 0"/>
   <geometry>
     <cylinder length="0.01" radius="0.01"/>
   </geometry>
 </visual>
 <collision>
   <origin xyz="0 0 -0.025" rpy="0 0 0"/>
   <geometry>
     <cylinder length="0.005" radius="0.01"/>
   </geometry>
 </collision>
</link>
<joint name="Sujec_camara" type="fixed" >
    <parent link="Base" />
    <origin xyz="0 0 -0.0" rpy="0 0 0.785398" />
    <child link="Camara" />
</joint>

<!--*****************GAZEBO PLUGINS****************-->
<!--**PLUGIN SENSOR DE ODOMETRIA**-->
<gazebo>
   <controller:gazebo_ros_p3d name="sensor_odometria" plugin="
       libgazebo_ros_p3d.so">
     <alwaysOn>true</alwaysOn> <!--ON/OFF-->
     <updateRate>10000</updateRate> <!--Frecuencia del topic-->
     <bodyName>Base</bodyName> <!--Cuerpo sobre el que se monta-->
     <frameName>plane</frameName> <!--Frame de referencia-->
     <topicName>Odometria</topicName> <!--Topic de salida-->
     <gaussianNoise>0.0</gaussianNoise> <!--Ruido en las medidas-->
   </controller:gazebo_ros_p3d>
</gazebo>
```

```
<!--**PLUGIN SENSOR CAMARA **-->
<gazebo reference="Camara">
   <material>Gazebo/Red</material>
   <sensor:camera name="Camara_sensor">
     <!--Resolucion-->
     <imageSize>640 480</imageSize>
     <!-- Distancia [m] a la que empieza a mostrar -->
     <nearClip>0.01</nearClip>
     <!-- Distancia a la que deja de mostrar-->
     <farClip>10</farClip>
     <controller:gazebo_ros_camera name="Camara_controller" plugin="
         libgazeb o_ros_camera.so">
       <alwaysOn>true</alwaysOn>
       <updateRate>100.0</updateRate>
       <imageTopicName>/Quadrotor/Camara/image_raw</imageTopicName>
       <cameraInfoTopicName>/Quadrotor/Camara/camera_info</
           cameraInfoTopicName>
       <frameName>Camara</frameName>
     </controller:gazebo_ros_camera>
   </sensor:camera>
</gazebo>

</robot>
```

B.2 display_quad_rviz.launch

```
<launch>
    <!-- Lectura robot como argumento al lanzar el roslaunch: ...model:=
        robot.urdf -->
    <arg name="model" default="$(find modelquad)/robots/
        quadrotor_continuous.urdf"/>

    <!-- Argumento para que joint_state_publisher se ejecute con interfaz
         grafica-->
    <arg name="gui" default="true" />

    <!--Para cargar en rviz un Robot, es necesario incluir su modelo como
         parametro "robot_description"-->
    <param name="robot_description" textfile="$(arg model)" />

    <!-- Equivalente para el argumento gui en joint_State_Publisher-->
    <param name="use_gui" value="$(arg gui)"/>

    <!-- Ejecutamos un nodo de j_s_p-->
    <node name="joint_state_publisher" pkg="joint_state_publisher" type="
        joint_ state_publisher" />

    <!-- Ejecutamos un nodo de r_s_p -->
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
        state_ publisher" />

    <!-- Ejecutamos rviz y cargamos la configuracion ____.vcg -->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find modelquad)/
        conf_rv iz/confrvizquadrotor.vcg" />
```

```
</launch>

<!-- PARA LANZARLO: $ roslaunch modelquad display_quad_rviz.launch-->
```

B.3 display_quad_gazebo.launch

```
<launch>
    <!--********** GAZEBO, URDF, Mesa y Quadrotor **********-->

    <!-- Parametro para usar tiempo de simulacion (/Clock)-->
    <param name="/use_sim_time" value="true" />

    <!--Ejecucion Gazebo con la configuracion empty.world-->
    <node name="gazebo" pkg="gazebo" type="gazebo" args="$(find
        gazebo_worlds)/worlds/empty.world" respawn="false" output="screen
        "/>

    <!--Cargamos el modelo de una mesa en el servidor de parmaetros-->
    <param name="table_description" command="$(find xacro)/xacro.py $(
        find modelquad)/robots/table.urdf.xacro" />

    <!--Enviamos la mesa a gazebo -->
    <node name="spawn_table" pkg="gazebo" type="spawn_model" args="-urdf
        -param table_description -z 0.01 -model table_model" respawn="
        false" output="screen" />

    <!--Cargamos el robot (Quad) en el servidor de parametros-->
    <param name="robot_gazebo_description" command="$(find xacro)/xacro.
        py $(find modelquad)/robots/quadrotor.urdf" />

    <!--Enviamos el robot a gazebo -->
    <node name="spawn_quad" pkg="gazebo" type="spawn_model" args="-urdf -
        param robot_gazebo_description -x 1 -y 0 -z 0.7 -model quad"
        respawn="false" output="screen" />

    <!--********** CAMERA ****************-->

    <!--Se ejecuta el visor pasandole el topic de la camara del Quad-->
    <node name="camara" pkg="image_view" type="image_view" args="image:=/
        Quadrotor/Camara/image_raw " respawn="false" output="screen">
        <param name="window_name" type="str" value="Camara Quadrotor" />
    </node>

    <!--********** RXPLOT (SENSOR DE ODOMETRIA) **********-->

    <!--Se ejecuta rxplot para el topic de Odometria (solo XYZ)-->
    <node pkg="rxtools" type="rxplot" name="rxplot_positionxyz" args="-r
        1 -l 'X [m]','Y [m]','Z [m]' -t PoscionXYZ /Odometria/pose/pose/
        position/x /Odometria/pose/pose/position/y /Odometria/pose/pose/
        position/z" />

</launch>
```

B.4 Propiedades_Quad.h

```
// Propiedades y constantes dinamicas y aerodinamicas
    #define g 9.800 // m/s2 Confirmado probando diferentes valores con
        aplicar_trottle_1_fza.cpp
    #define m 1.73 //kg
    #define Ixx 0.01632655 // kg m2 equivalente a 1 N m s2 en que lo
        expresan en elChapter 3
    #define Iyy 0.01632655 // kg m2
    #define Izz 0.02642555 // kg m2
    #define b 54.2e-6 // N s2
    #define d 1.1e-6 // N m s2
    #define l 0.286 // m
```

B.5 Controlador_Tustin.h

```
// Periodo de muestreo: inverso de la frecuencia del ciclo while{}
    #define T 0.1
// Coeficientes Controladores ZRPY
    #define kp_z 8
    #define Ti_z 40
    #define Td_z 4

    #define kp_r 20
    #define Ti_r 20
    #define Td_r 7.5

    #define kp_p 20
    #define Ti_p 20
    #define Td_p 7.5

    #define kp_y 20
    #define Ti_y 30
    #define Td_y 8

// COEFICIENTES CONTROLADORES X Y
    #define kp_X 0.12
    #define Ti_X 80
    #define Td_X 0.18

    #define kp_Y 0.12
    #define Ti_Y 80
    #define Td_Y 0.18
```

B.6 control.cpp

Nodo para controlar el quadrotor en X, Y, Z y *yaw*. Para obtener el control en *roll*, *pitch*, *yaw* y Z, simplemente ha de comentarse la parte señalada en el código referida y señalizada para el control en X e Y.

```
/// Incluye los funciones comunes de Ros y de STD (math.h,Services.h,..)
    #include "ros/ros.h"
```

```
/// Tipo de mensaje Servicio para la aplicacion de fuerzas (Gazebo)
    #include "gazebo_msgs/ApplyBodyWrench.h"
/// Tipo de mensaje de Odometria (common_msg)
    #include "nav_msgs/Odometry.h"
/// Libreria de bullet necesaria para transformar de Quaternion a RPY
    #include "LinearMath/btQuaternion.h"
    #include "LinearMath/btMatrix3x3.h"
/// Tipo de mensaje de Posicion + RPY (modelquad)
    #include "modelquad/Position_RPY.h"
/// Tipo de mensaje Servicio recepcion posiciones deseadas
    #include "modelquad/EnvioPosDeseada.h"
/// Tipo de mensaje de 4 datos para el giro de los motores (modelquad)
    #include "modelquad/Giros_Motores.h"
/// Coeficientes del PID discretizado por la aproximacion de Tustin
    #include "modelquad/Controlador_Tustin.h"
/// Propiedades dinamicas del quadrotor
    #include "modelquad/Propiedades_Quad.h"

// #define _USE_MATH_DEFINES //(PI) NO HACE FALTA YA ESTA EN ros.h
// #include "math.h"


/// Objetos de los tipo de mensaje de posicion actual y deseadas
/// Se declaran como variables globales para que sean accesibles desde
/// todas las funciones
    modelquad::Position_RPY pos_actual;
    modelquad::Position_RPY pos_deseada;

/// Flags para la espera inicial de una posicion deseada (y actual)
    int actual_ok=0;
    int deseada_ok=0;


/// Flag para limites dinamicos. A 1 si llego una nueva pos_deseada
    int cambio_deseada=0;
/// Limites dinamicos, se inicializan a 0.10 por ejemplo
    float limiteX=0.10;
    float limiteY=0.10;

/// Funcion lecturas topic posicion actual (sensor odometria)
/// Odom es un objeto del tipo Odometry en el que se guarda el mensaje
/// recibido por el topic /Odometria
void Calculo_posicion_actual(const nav_msgs::Odometry odom)
{
    //~ ROS_INFO("Entra actual");
    /// Variable para saber si esta girando en yaw a la izquierda(yaw>0)
    /// o a la derecha(yaw<0). Se declara static para que conserve su
    /// valor de una llamada a otra, sin volver a leerse esta linea
    static int a=0;
    /// POSICION (Respecto al sistema de referencia del plano (fijo))
    float pos_X=odom.pose.pose.position.x;
    float pos_Y=odom.pose.pose.position.y;
    float pos_z=odom.pose.pose.position.z;
    /// ORIENTACION (Respecto al sistema de referencia del QUAD (MOVIL))
    /// quaternion
```

```
    float qx=odom.pose.pose.orientation.x;
    float qy=odom.pose.pose.orientation.y;
    float qz=odom.pose.pose.orientation.z;
    float qw=odom.pose.pose.orientation.w;
    ROS_INFO("Orientation (qx,qy,qz,w):\t (%f,%f,%f)", qx, qy, qz, qw);
    /// Conversion quaternion a angulos de Euler (RPY).
    btQuaternion q(qx,qy,qz,qw);
    double roll, pitch, yaw; /// No vale float para getRPY()
    btMatrix3x3(q).getRPY(roll, pitch, yaw);
    /// Escalado del yaw tal que sea positivo de [0 a 2pi] y negativo
    /// de ]0 a -2pi]. El yaw leido por el sensor del quadrotor es
    /// positivo de [0 a pi] y negativo de ]pi a 2pi], lo cual provoca
    /// el cambio de signo en pi, dificultando el control
    if((yaw>M_PI/2)&&(yaw<M_PI)&&(a==0)) ///girando a la izquierda
        {
            a=1;
        }
    if((yaw<-M_PI/2)&&(yaw>-M_PI)&&(a==0)) ///girando a la derecha
        {
            a=2;
        }
    if ((yaw<0)&&(a==1)) ///girando a la izq +pi (convierto angulo en +)
        {
            yaw=yaw+2*M_PI;
        }
    if ((yaw>0)&&(a==2)) ///girando a la der -pi (convierto angulo en -)
        {
            yaw=yaw-2*M_PI;
        }
    if((yaw<M_PI/4)&&(yaw>-M_PI/4))///Reset (ojo: del yaw modificado)
        {
            a=0;
        }
    ROS_INFO("Orientation (r,p,y):\t (%f,%f,%f)",roll, pitch, yaw);
    /// Relleno del mensaje que se enviara por /Pos_rpy_actual
    /// Ademas al ser variable global son las que se utilizan como
    /// posiciones actuales en main()
    pos_actual.posicion.x=pos_X;
    pos_actual.posicion.y=pos_Y;
    pos_actual.posicion.z=pos_z;
    pos_actual.orientacion.x=roll;
    pos_actual.orientacion.y=pitch;
    pos_actual.orientacion.z=yaw;
    // ROS_INFO("\nPosicion (x,y,z):\t (%f,%f,%f)\nOrientation (r,p,y):\
        t (%f, %f,%f)", pos_X,pos_Y,pos_z,roll, pitch, yaw);
    // ROS_INFO("a: %d", a);
    /// Flag a 1
    actual_ok=1;
}

/// Función servicio recepcion de posiciones deseadas
bool Calculo_posicion_deseada(modelquad::EnvioPosDeseada::Request &req,
    modelquad::EnvioPosDeseada::Response &res)
{
    // ROS_INFO("Entra servicio pos_deseada");
```

```
    /// Se almacena en la variable global pos_deseada para ser accesible
    /// desde main()
    pos_deseada.posicion.x=req.pos_des.posicion.x;
    pos_deseada.posicion.y=req.pos_des.posicion.y;
    pos_deseada.posicion.z=req.pos_des.posicion.z;
    pos_deseada.orientacion.x=req.pos_des.orientacion.x;
    pos_deseada.orientacion.y=req.pos_des.orientacion.y;
    pos_deseada.orientacion.z=req.pos_des.orientacion.z;
    ROS_INFO("Posicion deseada (x,y,z)= (%f,%f,%f)", pos_deseada.
        posicion.x, pos_deseada.posicion.y, pos_deseada.posicion.z);
    /// Flag a 1
    deseada_ok=1;
    /// Flag a 1
    cambio_deseada=1;
    /// Rspuesta servicio
    res.confirmacion=true;
    return true;
}


int main(int argc, char **argv)
{
        ROS_INFO("Valor de pi utilizado: %f",M_PI);
    /// VARIABLES LOCALES:
    /// posicion actual y anteriores
        float Xk_T=0;
        float Xk_T_d=0;
        float Xk_Q_d=0;
        float Yk_T=0;
        float Yk_T_d=0;
        float Yk_Q_d=0;
        float Xk_T_g=0;
        float Yk_T_g=0;
        float Xk_1=0;
        float Xk_2=0;
        float Yk_1=0;
        float Yk_2=0;
        float zk=0;
        float zk_1=0;
        float zk_2=0;
        float rk=0;
        float rk_1=0;
        float rk_2=0;
        float pk=0;
        float pk_1=0;
        float pk_2=0;
        float yk=0;
        float yk_1=0;
        float yk_2=0;
    /// Terminos integrales y derivativos PIDs
        float derivadaX=0;
        float derivadaY=0;
        float integralX=0;
        float integralY=0;
        float integralz=0;
```

```
    float integralr=0;
    float integralp=0;
    float integraly=0;
/// Accion de control en aceleracion
    float uXk, uYk, uzk, urk, upk, uyk;
/// Error actual
    float eXk, eYk, ezk, erk, epk, eyk;
/// Errores en z,r,p,y anteriores
    float eXk_1=0;
    float eXk_2=0;
    float eYk_1=0;
    float eYk_2=0;
    float ezk_1=0;
    float ezk_2=0;
    float erk_1=0;
    float erk_2=0;
    float epk_1=0;
    float epk_2=0;
    float eyk_1=0;
    float eyk_2=0;
/// Accion de control paso anterior
    float uXk_1=0;
    float uYk_1=0;
    float uzk_1=0;
    float urk_1=0;
    float upk_1=0;
    float uyk_1=0;
/// Fuerzas y momentos incrementales (los que se envian a Gazebo)
    float U1, U2, U3, U4;
/// Fuerzas y momentos anteriores (el que se esta aplicando)
    float U1ant=0;
    float U2ant=0;
    float U3ant=0;
    float U4ant=0;
/// Fuerzas y momentos actuales (Total a aplicar)
    float U1act=0;
    float U2act=0;
    float U3act=0;
    float U4act=0;
/// Coeficientes parametros aerodinamicos
    float A = 1.0/4/b;
    float B = 1.0/2/b/l;
    float C = 1.0/4/d;

/// CREACION NODO CON SU NOMBRE Y ETIQUETA:
    ros::init(argc, argv, "control"); ros::NodeHandle n;

/// Espera a que empieze a emitir /clock
    ros::Time::waitForValid();

/// SERVICIO APLICACION DE FUERZAS:
/// Objetos de la clase de gazebo::ApplyBodyWrench para rellenar y
/// enviar por servicio de aplicacion de fuerzas (Gazebo)
    gazebo_msgs::ApplyBodyWrench srv1;
```

```
/// Rellenamos las opciones que no variaran (Todas a excepcion de Fz
    y Momentos
/// y el instante de aplicacion de aplicacion)
    srv1.request.body_name = "quad::Base";
    srv1.request.reference_frame= "quad::Base";
    srv1.request.reference_point.x=0;
    srv1.request.reference_point.y=0;
    srv1.request.reference_point.z=0;
    srv1.request.wrench.force.x=0;
    srv1.request.wrench.force.y=0;
    srv1.request.wrench.force.z=0;
    srv1.request.wrench.torque.x=0;
    srv1.request.wrench.torque.y=0;
    srv1.request.wrench.torque.z=0;
    srv1.request.duration = ros::Duration(1200);
///Nota: La idea es que la fuerza se apliquen durante 20 minutos,
///siendo todas a excepcion de la primera incrementos. Realmente para
///simular la bateria habria que ir disminuyendo esa duracion confor-
///me pasa el tiempo de simulacion, que de momento no se esta
    haciendo

/// Objeto a rellenar con el error en posicion y orientacion
    modelquad::Position_RPY error;
///Objeto a rellenar con los giro de los motores
    modelquad::Giros_Motores giros;

/// TOPICS:
/// Topic por el que se reciben los mensajes del sensor de odometria
    ros::Subscriber Sub_pos_actual = n.subscribe("/Odometria", 1,
        Calculo_posicio n_actual);
/// Topic por el que se enviaran la posicion y RPY actuales
    ros::Publisher Pub_pos_actual = n.advertise<modelquad::
        Position_RPY>("Pos_RPY _actual", 100);
/// Topic por el que se enviaran los giros de los motores
    ros::Publisher Pub_giros_motores = n.advertise<modelquad::
        Giros_Motores>("Giros_Motores", 100);

/// SERVICIOS:
/// Creacion servicio de recepcion de la posicion deseada
    ros::ServiceServer service = n.advertiseService("
        envio_pos_deseada", Calculo _posicion_deseada);
/// Cliente al servicio de aplicacion de fuerzas (Gazebo)
    ros::ServiceClient client = n.serviceClient<gazebo_msgs::
        ApplyBodyWrench>("/gazebo/apply_body_wrench");

/// Se duerme durante 5 segundos para que todo este en orden en
    Gazebo
ros::Duration(5,0).sleep();

/// Espera activa hasta primera lectura de posiciones actual y
/// deseada para no empezar el control y permanecer en la posicion
/// inicial mientras que el controlador estaria integrando errores
/// incorrectos
    while((actual_ok==0)||(deseada_ok==0))
    {
```

```
        /// Lecturas posicion actual
        ros::spinOnce();
        /// Se duerme 10 microseg para no sobrecargar la CPU
        ros::Duration(0,10000).sleep();
    }

/// Condiciones iniciales -- Antes de despegue
    zk_1=pos_actual.posicion.z;
    zk_2=pos_actual.posicion.z;

/// Frecuencia del ciclo (while) 10 Hz, necesario loop.sleep()
    ros::Rate loop(10);

/// BUCLE PRINCIPAL
while(ros::ok())
{
    ///Llamada a todos los mensajes que esten en cola
        ros::spinOnce();

    /// Publicacion en topic de la posicion actual
        Pub_pos_actual.publish(pos_actual);

    /// Posicion actual del quadrotor respecto al
    /// SR de tierra
        Xk_T = pos_actual.posicion.x;
        Yk_T = pos_actual.posicion.y;
        zk = pos_actual.posicion.z;
    /// SR del Quad
        rk = pos_actual.orientacion.x;
        pk = pos_actual.orientacion.y;
        yk = pos_actual.orientacion.z;

    /// Posicion deseada del quadrotor respecto al SR de Tierra
        Xk_T_d = pos_deseada.posicion.x;
        Yk_T_d = pos_deseada.posicion.y;

    /// Transformacion de las posiciones en x e y deseadas al SR Quad
    /// (Cambio SR de tierra al del quadrotor)
        Xk_Q_d = Xk_T_d * cos(yk) + Yk_T_d * sin(yk) - Yk_T * sin(yk)
            - Xk_T * cos(yk);
        Yk_Q_d = -Xk_T_d * sin(yk) + Yk_T_d * cos(yk) - Yk_T * cos(yk)
             + Xk_T * sin(yk);

    /// Posicion actual del quadrotor respecto al SR de tierra pero
    /// girado tal que su x e y son paralelos al del quadrotor
    /// (necesario para hacer la derivada de la referencia en el PID)
        Xk_T_g = Xk_T * cos(yk) + Yk_T * sin(yk);
        Yk_T_g = -Xk_T * sin(yk) + Yk_T * cos(yk);

    //ROS_INFO("X_Q_d: %f, Y_Q_d: %f, X_T_g: %f, Y_T_g: %f", Xk_Q_d,
         Yk_Q_d, X k_T_g, Yk_T_g);//COMPROBADO

    /// Limites dinamicos:
    if(cambio_deseada==1)
    {
```

```
    ///Limites en Roll y pitch deseados proporicionales al error.
    ///Siendo esta constante de proporcionalidad el limite cuando
    ///el error es de un metro debido a que se hizo asi la
    ///sintonizacion
        limiteX=0.1*std::abs(Xk_Q_d);
        limiteY=0.1*std::abs(Yk_Q_d);
    ///Limites minimos: Casos en que apenas varie x o y
    ///Nunca deben ser cero pues no se contrarrestarian perturbac.
        if((limiteX<0.001)&&(limiteX>0)){limiteX=0.001;}
        if((limiteY<0.001)&&(limiteY>0)){limiteY=0.001;}
    ///Limites maximos: Casos en que se desee un incremento en X e
    ///Y mayor de 2 m. Un incremento deseado de 2m nos lleva a un
    ///limite de 0.2 rad(unos 11.5 grados). Graficamente se mide
    ///una velocidad de 1.108 m/s
        if(limiteX>0.2){limiteX=0.2;}
        if(limiteY>0.2){limiteY=0.2;}
        ROS_INFO("limiteX: %f limiteY: %f", limiteX, limiteY);
        cambio_deseada=0;
}

/// ******************CONTROL EN XY*******************///
///COMENTANDO TODO SE PASA A CONTROLAR DIRECTAMENTE EN
///ROLL Y EN PITCH
// Posiciones en SR del quad son directamnte el error cometido
    error.posicion.x= Xk_Q_d;
    error.posicion.y= Yk_Q_d;
    eXk=error.posicion.x;
    eYk=error.posicion.y;
    integralX=(eXk+eXk_1);
    integralY=(eYk+eYk_1);
    derivadaX=-(Xk_T_g - 2*Xk_1 + Xk_2);
    derivadaY=-(Yk_T_g - 2*Yk_1 + Yk_2);
///PIDs X e Y --> Roll y Pitch deseados
    uXk = uXk_1 + kp_X*(eXk-eXk_1) + (T/(2*Ti_X))*integralX + (
        Td_X/T)*derivadaX;
    uYk = uYk_1 + kp_Y*(eYk-eYk_1) + (T/(2*Ti_Y))*integralY + (
        Td_Y/T)*derivadaY;
/// Limites en roll y pitch
    if(uXk>limiteX){uXk=limiteX;}
    if(uXk<-limiteX){uXk=-limiteX;}
    if(uYk>limiteY){uYk=limiteY;}
    if(uYk<-limiteY){uYk=-limiteY;}
/// Ref del roll (Y --> -Roll)
    pos_deseada.orientacion.x = - uYk;
/// Ref del pith (X --> Pitch)
    pos_deseada.orientacion.y = uXk;
// ROS_INFO("X: Proporcional: %f, Integral= %f, Derivada= %f",
    kp_X*(eXk-eXk_1), (T/(2*Ti_X))*integralX, (Td_X/T)*derivadaX);
// ROS_INFO("Y: Proporcional: %f, Integral= %f, Derivada= %f",
    kp_Y*(eYk-eYk_1), (T/(2*Ti_Y))*integralY, (Td_Y/T)*derivadaY);
// ROS_INFO("Roll deseado: %f, Pitch deseado: %f", pos_deseada.
    orientacion.x, pos_deseada.orientacion.y);
/// ****************FIN CONTROL XY*******************///

/// Calculo del error
```

```
    error.posicion.z= pos_deseada.posicion.z - pos_actual.posicion
        .z;
    error.orientacion.x = pos_deseada.orientacion.x - pos_actual.
        orientacion.x;
    error.orientacion.y = pos_deseada.orientacion.y - pos_actual.
        orientacion.y;
    error.orientacion.z = pos_deseada.orientacion.z - pos_actual.
        orientacion.z;
///Nota: Se rellena primero como mensaje Pos_RPY por si es
///necesario mas tarde publicarlo

/// Solo para facilitar escritura
    ezk=error.posicion.z; //z
    erk=error.orientacion.x; //r
    epk=error.orientacion.y; //p
    eyk=error.orientacion.z; //y

/// PIDs Z, roll, pitch y yaw
    integralz=(ezk+ezk_1);
    integralr=(erk+erk_1);
    integralp=(epk+epk_1);
    integraly=(eyk+eyk_1);
    uzk = uzk_1 + kp_z*(ezk-ezk_1) + (T/(2*Ti_z))*integralz - (
        Td_z/T)*(zk - 2*zk_1 + zk_2);
    urk = urk_1 + kp_r*(erk-erk_1) + (T/(2*Ti_r))*integralr - (
        Td_r/T)*(rk - 2*rk_1 + rk_2);
    upk = upk_1 + kp_p*(epk-epk_1) + (T/(2*Ti_p))*integralp - (
        Td_p/T)*(pk - 2*pk_1 + pk_2);
    uyk = uyk_1 + kp_y*(eyk-eyk_1) + (T/(2*Ti_y))*integraly - (
        Td_y/T)*(yk - 2*yk_1 + yk_2);

/// Modelo Inverso (aceleraciones-->Fuerzas y momentos)
/// Fuerza[N]
    U1act = m*(uzk + g)/(cos(rk)*cos(pk));
/// Momentos[N m]
    U2act = urk*Ixx;
    U3act = upk*Iyy;
    U4act = uyk*Izz;
///Nota: Serian las fuerzas actuales a aplicar, pero a Gazebo se
    le
///envian incrementales segun la forma escogida
    // ROS_INFO("Uact: %f,%f,%f,%f",U1act,U2act,U3act,U4act);
/// Fuerzas Incrementales respecto a la anterior
    U1=U1act-U1ant;
    U2=U2act-U2ant;
    U3=U3act-U3ant;
    U4=U4act-U4ant;
    // ROS_INFO("U: %f,%f,%f,%f",U1,U2,U3,U4);

/// Relleno servicio: Fuerzas y momentos a enviar
    srv1.request.wrench.force.z=U1;
    srv1.request.wrench.torque.x=U2;
    srv1.request.wrench.torque.y=U3;
    srv1.request.wrench.torque.z=U4;
/// Instante de aplicacion de las fuerzas
```

```
    ros::Time ahora = ros::Time::now();
    srv1.request.start_time = ahora;
///Nota: se deja asi por si se quiere insertar un retraso, pues
///podria resultar inestable la aplicacion en un instante
    anterior
///al actual de simulacion
/// LLAMADA AL SERVICIO APLICACION DE FUERZA (Gazebo)
    client.call(srv1);
    // ROS_INFO("FUERZAS[N]: (Fz,Mx,My,Mz)= (%f, %f, %f, %f)", U1
        , U2, U3, U4);

/// CALCULO GIRO MOTORES
    giros.G1 = sqrt( A*U1act - B*U3act - C*U4act);
    giros.G2 = sqrt( A*U1act - B*U2act + C*U4act);
    giros.G3 = sqrt( A*U1act + B*U3act - C*U4act);
    giros.G4 = sqrt( A*U1act + B*U2act + C*U4act);
/// Publicacion por el topic
    Pub_giros_motores.publish(giros);

/// ACTUALIZACIONES DE VARIABLES
    Xk_2=Xk_1;
    Xk_1=Xk_T_g;
    Yk_2=Yk_1;
    Yk_1=Yk_T_g;
    zk_2=zk_1;
    zk_1=zk;
    rk_2=rk_1;
    rk_1=rk;
    pk_2=pk_1;
    pk_1=pk;
    yk_2=yk_1;
    yk_1=yk;

    uXk_1=uXk;
    uYk_1=uYk;
    uzk_1=uzk;
    urk_1=urk;
    upk_1=upk;
    uyk_1=uyk;

    eXk_2=eXk_1;
    eYk_2=eYk_1;
    ezk_2=ezk_1;
    erk_2=erk_1;
    epk_2=epk_1;
    eyk_2=eyk_1;
    eXk_1=eXk;
    eYk_1=eYk;
    ezk_1=ezk;
    erk_1=erk;
    epk_1=epk;
    eyk_1=eyk;

    U1ant=U1act;
    U2ant=U2act;
```

```
            U3ant=U3act;
            U4ant=U4act;

        /// Espera fin de ciclo (10 Hz)
            // ROS_INFO("Comienza sleep");
            loop.sleep();
            // ROS_INFO("Fin sleep");
    }
    return 0;
}
```

B.7 quadrotor_control.launch

```
<launch>
    <!--********** GAZEBO, URDF **********-->
        <!-- Usar tiempo de simulacion en vez del real-->
        <param name="/use_sim_time" value="true" />

        <!--Lanza Gazebo en mundo vacio-->
        <node name="gazebo" pkg="gazebo" type="gazebo" args="-u $(find
            gazebo_worlds)/worlds/empty.world" respawn="false" output="log
            "/>

        <!-- Enviamos el robot (Quad) al servidor de parametros para que
            lo lea Gazebo -->
        <param name="robot_gazebo_description" command="$(find xacro)/
            xacro.py $(find modelquad)/robots/quadrotor.urdf" />

        <!-- Lanzamos el robot (Quad) con el nodo spawn_model de Gazebo.
            Los parametros son el nombre en el servidor de parametros, la
            localizacion inicial en Gazebo, ... -->
        <node name="spawn_quad" pkg="gazebo" type="spawn_model" args="-
            urdf -param robot_gazebo_description -x 0 -y 0 -z 0.7 -model
            quad" respawn="false" output="screen" />

    <!--********** NODO CONTROL **********-->
    <node name="control" pkg="modelquad" type="control" output="screen"/>

    <!--********** CAMERA ****************-->
    <node name="camara" pkg="image_view" type="image_view" args="image:=/
        Quadrotor/Camara/image_raw " respawn="false" output="screen">
        <param name="window_name" type="str" value="Camara Quadrotor" />
    </node>

    <!--********** RXCONSOLE **********-->
    <node pkg="rxtools" type="rxconsole" name="rxconsole" />

    <!--********** DINAMIC RECONFIGURE **********-->
        <!--<node pkg="dynamic_reconfigure" type="reconfigure_gui" name="
            reconfigure_gui" />-->
        <!-- PROGRAMA PARA VARIAR LOS PARAMETROS,EN ESTE CASO, DE GAZEBO
            -->
```

```
    <!--********** RXPLOT ***********-->
    <node pkg="rxtools" type="rxplot" name="rxplot_positionxyz" args="-r
        1 -l 'X[m]','Y [m]','Z[m]' -t PoscionXYZ /Odometria/pose/pose/
        position/x /Odometria/pose/pose/position/y /Odometria/pose/pose/
        position/z" />

    <node pkg="rxtools" type="rxplot" name="rxplot_giros_motores" args="-
        r 1 -l 'G1[rad/s]','G2[rad/s]','G3 [rad/s]','G4[rad/s]' -t
        Giros_Motores /Giros_Motores/G1 /Giros_Motores/G2 /Giros_Motores/
        G3 /Giros_Motores/G4" />

    <!-- Otros posibles plot:
    <node pkg="rxtools" type="rxplot" name="rxplot_orientation" args="-r
        1 -l 'Roll [rad]','Pitch [rad]','Yaw [rad]' -t Orientacion /
        Pos_RPY_actual/orientacion/x /Pos_RPY_actual/orientacion/y /
        Pos_RPY_actual/orientacion/z" />

    (...) -->
</launch>
```

B.8 arranque.cpp

Se adjunta *arranque.cpp* como nodo tipo para el envío de posiciones deseadas. Todos los demás (*Z1X1.cpp*, *Z1r01.cpp*, ...) son equivalentes a este, solo variando la posición y/o la orientación enviada.

```
#include "ros/ros.h"
/// Tipo de mensaje de Posicion + RPY (modelquad)
    #include "modelquad/Position_RPY.h"
/// Tipo mensaje del servicio para enviar puntos al control
    #include "modelquad/EnvioPosDeseada.h"

int main(int argc, char **argv) {
/// Creacion del nodo con su nombre y etiqueta nodehandle
    ros::init(argc, argv, "navegacion");
    ros::NodeHandle n4;
/// Espera a que empieze a emitir /clock
    ros::Time::waitForValid();
/// Cliente al servicio EnvioPosDeseada:
    ros::ServiceClient client =n4.serviceClient<modelquad::
        EnvioPosDeseada>("envio_pos_deseada");
    modelquad::EnvioPosDeseada srv;

/// Envio posicion actual en el suelo para simular arranque motores
/// (Solo en arranque.cpp)
    srv.request.pos_des.posicion.x=0;
    srv.request.pos_des.posicion.y=0;
    srv.request.pos_des.posicion.z=0.4900999; //Altura CM en suelo
    srv.request.pos_des.orientacion.x=0;
    srv.request.pos_des.orientacion.y=0;
    srv.request.pos_des.orientacion.z=0;
    if (client.call(srv)) //Llamada al servicio
```

```
    {
        ROS_INFO("Arranque motores correctos");
    }
    else
    {
        ROS_INFO("Error en la llamada al servicio durante el arranque");
    }

ros::Duration(10,0).sleep();

/// Despegue a 1 metro
    srv.request.pos_des.posicion.x=0;
    srv.request.pos_des.posicion.y=0;
    srv.request.pos_des.posicion.z=1;
    srv.request.pos_des.orientacion.x=0;
    srv.request.pos_des.orientacion.y=0;
    srv.request.pos_des.orientacion.z=0;
    if (client.call(srv)) //Llamada al servicio
    {
        ROS_INFO("Ascenso a un metro correcto");
    }
    else
    {
        ROS_INFO("Error en la llamada al servicio durante el ascenso a un
             metro");
    }
return 0;
}
```

B.9 gen_trayectoria.cpp

```
/// Incluye los funciones comunes de Ros y de STD (math.h, Services.h
    ,....)
    #include "ros/ros.h"
/// Incluimos archivo de cabecera en el que se define la trayectoria a
    seguir
    #include "modelquad/trayectoria.h"
/// Tipo de mensaje de Odometria (common_ msg)
    #include "nav_msgs/Odometry.h"
/// Tipo de mensaje para representacion de posicion (x,y,z)
    #include "geometry_msgs/Point.h"
/// Tipo mensaje del servicio para envia r puntos al control
    #include "modelquad/EnvioPosDeseada.h"

/// Posicion actual del quadrotor. Global para poder ser llamada desde
    main y posicion actual
    geometry_msgs::Point pos_actual; // Point=(x,y,z)

/// Funcion Topic /Odometria

void posicion_actual(const nav_msgs::Odometry odom)
{
```

```
    /// POSICION (Respecto al sistema de referencia del plano (fijo))
    pos_actual = odom.pose.pose.position;
}

///Funcion auxiliar: devuelve el radio equivalente
float radio_equiv(float x, float y)
{
    float result = sqrt( pow( x - pos_actual.x , 2) + pow( y - pos_actual
        .y, 2) );
    return result;
}

int main(int argc, char **argv)
{
    ROS_INFO("Entra generador de trayectorias");

    /// Creacion del nodo con su nombre y etiqueta nodehandle
    ros::init(argc, argv, "gen_trayectoria");
    ros::NodeHandle ng;

    /// Espera a que empieze a emitir /clock (PLAY GAZEBO)
    ros::Time::waitForValid();

    /// Subscripcion al topic por el que se reciben los mensajes del
        sensor de odometria del quadrotor.urdf
    ros::Subscriber Sub_pos_actual = ng.subscribe("/Odometria", 1,
        posicion_actual);

    /// Cliente al servicio EnvioPosDeseada:
    ros::ServiceClient client =ng.serviceClient<modelquad::
        EnvioPosDeseada>("envio_pos_deseada");
    modelquad::EnvioPosDeseada srv;

    /// Publicamos por un topic los puntos de la trayectoria enviados al
        control para poder sacarlos por grafica
    ros::Publisher Pub_pos_comandada = ng.advertise<geometry_msgs::Point
        >("gen_trayectoria/pos_comandada", 1000);
    geometry_msgs::Point pos_comandada;

    /// Tamanio del vector que contiene la trayectoria
    int n=sizeof(tray_x)/sizeof(float);

    /// Frecuencia del ciclo (while) a 10 Hz, necesario incluir loop.
        sleep()
    ros::Rate loop(1);

    /// Indice al final de la iteracion
    int k = 0;

    while(ros::ok())
    {
        /// Lectura de la posicion actual = Llamada a la funcion
            posicion_actual
        ros::spinOnce();
```

```
/// Calculo punto de la trayectoria a enviar al control de
    posicion del quad

///El indice de busqueda comienza a partir del punto donde se
    quedo la ultima iteracion
int i = k;
float R = 0;

/// Calculo nuevo punto a enviar
/// En el caso de que el indice exceda el maximo (n) sera la
    ultima componente (n+1)

do{
       R = radio_equiv( tray_x[i], tray_y[i]);
       i = i+1;
} while( (R <= L) && (i < n) );

/// guardamos indice del punto que se envia para partir de el en
    la siguiente busqueda
k = i-1;

/// Una vez que salgamos del do_while i-1 sera el indice del
    punto de l
a trayectoria mas alejado del quadrotor a una distancia menor que
     R a partir de
l punto de la trayectoria que enviamos en el ciclo anterior

/// Enviamos el punto al control de posicion mediante llamada al
    servicio
srv.request.pos_des.posicion.x = tray_x[i-1];
srv.request.pos_des.posicion.y = tray_y[i-1];
srv.request.pos_des.posicion.z = tray_z[i-1];
srv.request.pos_des.orientacion.x = 0;
srv.request.pos_des.orientacion.y = 0;
srv.request.pos_des.orientacion.z = 0;

if (client.call(srv)) ///Llamada al servicio
{
    ROS_INFO("Llamada correcta al control\nPosicion enviada: [%f,
        %f, %f]", srv.request.pos_des.posicion.x, srv.request.
        pos_des.posicion.y, srv.request.pos_des.posicion.z);
}
else
{
    ROS_INFO("Error en la llamada al servicio del control del
        quadrotor");
}
/// Caso fin de trayectoria
if(i==n)
{
    ROS_INFO("Fin de la trayectoria");
    return 0;
}

/// Publicacion por el topic de la posicion comandada
```

```
        pos_comandada.x = tray_x[i-1];
        pos_comandada.y = tray_y[i-1];
        pos_comandada.z = tray_z[i-1];
        Pub_pos_comandada.publish(pos_comandada);

        loop.sleep(); // Espera fin de ciclo (10 Hz)
    }
    return 0;
}
```

B.10 seguimiento_trayectoria.launch

```
<launch>
    <!-- Usar tiempo de simulacion en vez del real-->
    <param name="/use_sim_time" value="true" />

    <!--Lanza Gazebo en mundo vacio-->
    <node name="gazebo" pkg="gazebo" type="gazebo" args="-u $(find
        gazebo_worlds)/worlds/empty.world" respawn="false" output="log"/>

    <!-- Enviamos el robot (Quad) al servidor de parametros para que lo
        lea Gazebo -->
    <param name="robot_gazebo_description" command="$(find xacro)/xacro.
        py $(find modelquad)/robots/quadrotor.urdf" />

    <!-- Lanzamos el robot (Quad) con el nodo spawn_model de Gazebo. Los
        parametros son el nombre en el servidor de parametros, la
        localizacioninicial en Gazebo, ... -->
    <node name="spawn_quad" pkg="gazebo" type="spawn_model" args="-urdf -
        param robot_gazebo_description -x 0 -y 0 -z 0.7 -model quad"
        respawn="false" output="screen" />

    <!-- Ejecutamos el control del quadrotor-->
    <node name="control" pkg="modelquad" type="control" output="screen"/>

    <!-- Ejecutamos el generador de trayectorias -->
    <node name="gen_trayectoria" pkg="modelquad" type="gen_trayectoria"
        output="screen"/>

    <!-- Monitorizacion -->
    <node pkg="rxtools" type="rxconsole" name="rxconsole" />

    <!-- Camara-->
    <node name="camara" pkg="image_view" type="image_view" args="image:=/
        Quadrotor/Camara/image_raw " respawn="false" output="screen">
        <param name="window_name" type="str" value="Camara Quadrotor" />
    </node>

    <!-- Grafica conjunta de XYZ actual del quadrotor y la referencia
        enviada-->
    <node pkg="rxtools" type="rxplot" name="rxplot_positionxyz" args="-r
        1 -l 'X[m]','Xref','Y [m]','Yref','Z[m]','Zref' -t PoscionXYZ '/
        Odometria/pose/pose/position/x','gen_trayectoria/pos_comandada/x'
```

```
        '/Odometria/pose/pose/position/y','gen_trayectoria/pos_comandada
        /y' '/Odometria/pose/pose/position/z','gen_trayectoria/
        pos_comandada/z'" />

    <!-- Grafica velocidad de giro de los motores-->
    <node pkg="rxtools" type="rxplot" name="rxplot_giros_motores" args="-
        r 1 -l 'G1[rad/s]','G2[rad/s]','G3 [rad/s]','G4[rad/s]' -t
        Giros_Motores /Giros_Motores/G1 /Giros_Motores/G2 /Giros_Motores/
        G3 /Giros_Motores/G4" />
</launch>
```

Apéndice C

Sintonización PIDs QuadRotor

C.1 PID en altura

La sintonización se hace en Matlab/Simulink mediante el modelo del quadrotor en altura dado por la primera ecuación del modelo del sistema de ecuaciones (2.8) suponiendo un *roll* y un *pitch* pequeños de forma que sus cosenos valgan la unidad. Luego se prueba y ajusta en Gazebo con nuestro modelo 3D de simulación y nuestro programa de control.

El método utilizado es el de Ziegler-Nichols en bucle cerrado, ya que el sistema en bucle abierto es inestable. Se procede por tanto buscando la ganancia crítica (K_C) del sistema controlado solamente mediante acción de control proporcional, para el cual la se debe obtener una respuesta con oscilaciones mantenidas. La ganancia K_p para la que se consiga es la ganancia K_C.

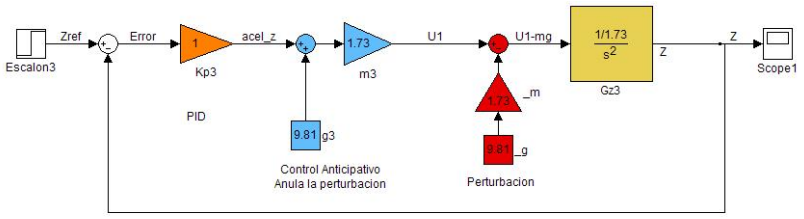


Figura C.1: Sistema Z-N en BC para Z

La ganancia proporcional se muestra en naranja, la acción de control anticipativa del efecto de la gravedad (perteneciente al modelo invertido que se utiliza en el control) en azul, la perturbación de la gravedad (ya que aparecerá en Gazebo) en rojo y el resto de la dinámica en la función de transferencia en amarillo.

Se obtiene una respuesta con oscilaciones mantenidas para todo K_p, lo cual permite usar el método aun que el controlador PID que se obtenga no será próximo al mejor que se pueda obtener. Nos quedamos con K_p = 1, para la cual la respuesta es la que se muestra en la Figura C.2.

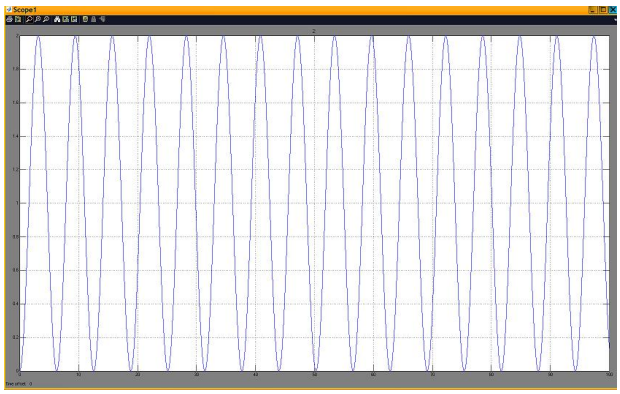


Figura C.2: Oscilaciones mantenidas en Z

Por tanto, el valor de la ganancia crítica es $K_C = K_p = 1$ y el valor del periodo crítico es $T_C = 6{,}283$ segundos, medido gráficamente. Para obtener los valores de los coeficientes del PID se utiliza la tabla del método[17]:

Tabla C.1: Método de Z-N en bucle cerrado

Tipo	K_p	T_i	T_d
P	$0{,}5\,K_C$	–	–
PI	$0{,}45\,K_C$	$T_C/1{,}2$	$T_C/1{,}2$
PID	$0{,}6\,K_C$	$T_C/2$	$0{,}125\,T_C$

Resultando:

$$\begin{cases} K_p = 0{,}6 \\ T_i = 3{,}1415 \\ T_d = 0{,}785 \end{cases} \tag{C.1}$$

Es importante recalcar que la estructura del PID que se obtiene por este método es no interactiva, pero también no paralela, lo que significa que la K_p multiplica a las tres acciones de control. Modificando el controlador en Simulink al PID con estos valores:

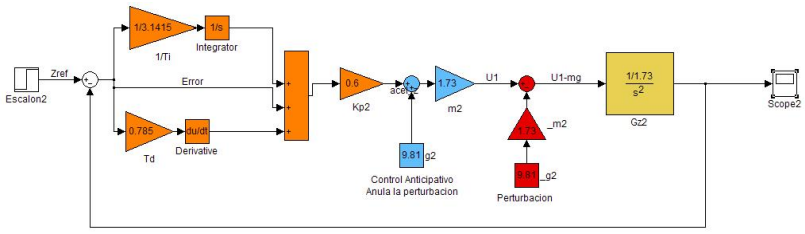


Figura C.3: Sistema controlado con PID de Z-N

La respuesta en altura que se obtiene para un escalón unitario en la

entrada es estable pero con unas pobres especificaciones de la respuesta transitoria, presentando un alto número de sobreoscilaciones, una SO de aproximadamente el 80 %, un tiempo de subida de 2.1 segundos y un tiempo de establecimiento de aproximadamente 90 segundos:

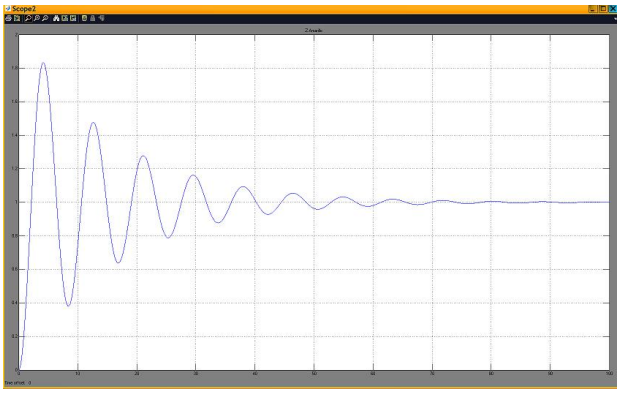


Figura C.4: Respuesta en Z para PID de Z-N

Antes de realizar el ajuste fino, se pasa el PID a su forma en paralelo, en el que las tres acciones son independientes una de la otra y es más sencillo de sintonizar manualmente:

$$\begin{cases} K_p = 0{,}6 \\ T_i = 5{,}236 \\ T_d = 0{,}471 \end{cases} \tag{C.2}$$

Para mejorarla, se modifican los coeficientes del PID teniendo en cuenta las siguientes ideas generales:

1. Aumentar la acción proporcional, aumentando K_p, hace que el sistema sea más estable y disminuye el tiempo de subida, pero también nos aumenta la SO y el número de ellas.

2. Aumentar la acción integral, disminuyendo T_i, empeora el transitorio en todos los sentidos, pero mejora el rechazo de perturbaciones y hace que el sistema consiga más rápidamente que el error en régimen permanente sea nulo.

3. Aumentar la acción derivativa, aumentando T_d, mejora el transitorio disminuyendo el tiempo de establecimiento y las sobreoscilaciones. Sin embargo no mejora el permanente y si se aumenta demasiado puede incluso desestabilizar el sistema.

Además se sustituye la derivada del error por la de la referencia debido a que se obtenía unos impulsos muy elevados en ella.

Tras un buen número de simulaciones, se llega a varios posibles PIDs, cuyas respuestas pueden considerarse aceptables. Se van a comentar los dos

que luego se probarán en ROS y veremos que solo uno sigue siendo bueno en este simulador.

El primero, que sería el más adecuado, tendría los siguientes coeficientes, estructura y respuesta en altura:

$$\begin{cases} K_p = 20 \\ T_i = 30 \\ T_d = 10 \end{cases} \tag{C.3}$$

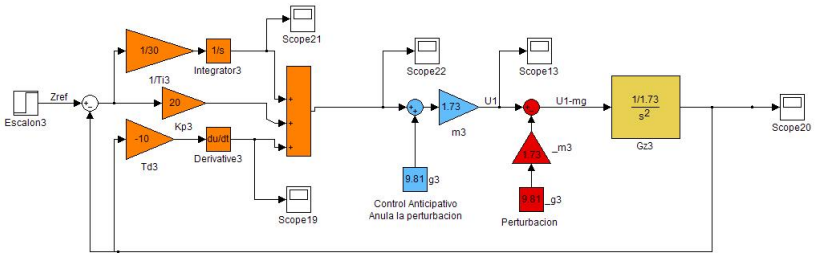


Figura C.5: Sistema controlado con PID paralelo

Se obtiene una respuesta no sobreoscilante con tiempo de subida y establecimiento (al 99,5 %) de 2 segundos, tal y como puede comprobarse en la Figura C.6.

Figura C.6: Respuesta en Z primera sintonización

Sin embargo este presenta un comportamiento oscilante al pasarlo al simulador en ROS, es decir, al discretizarlo y controlar un sistema equivalente pero tridimensional. Por lo cual también se hace otra sintonización que tendría la misma estructura pero los siguientes valores de los coeficientes del PID:

$$\begin{cases} K_p = 8 \\ T_i = 40 \\ T_d = 4 \end{cases} \tag{C.4}$$

Para este, la respuesta en Matlab si es sobreoscilante, pero por lo demás es aproximadamente igual:

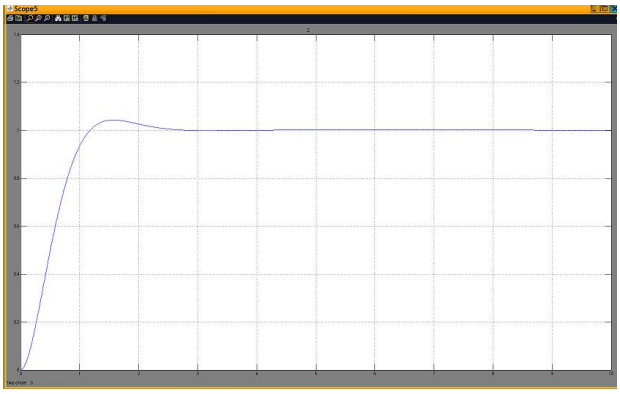


Figura C.7: Respuesta en Z segunda sintonización

Las especificaciones de la respuesta temporal en este caso son un 4.3 % de S.O, un tiempo de subida de 1.17 segundos y un tiempo de establecimiento (al 99.5 %) de 2.5 segundos.

En este segundo PID, que es el que se usa en las simulaciones en ROS/-Gazebo, se ha disminuido la acción derivativa que como se verá en dichas simulaciones es debido a el rizado que aparece debido a un Td tan alto.

C.2 PID en roll, pitch y yaw

Debido a utilizar la inversión del modelo (de cada ecuación) sobre la actuación, las respuestas obtenidas en Matlab para estos tres ángulos será la misma, ya que realmente la función de transferencia que nos quedaría en todos es un doble integrador. Si se realiza el método de Z-N en bucle cerrado, por ejemplo, para el roll:

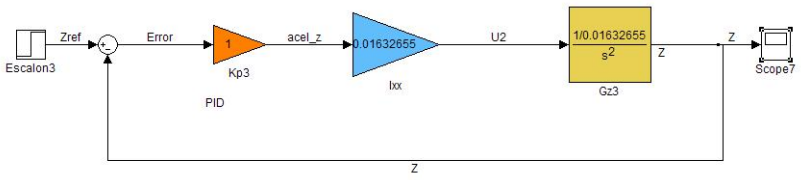


Figura C.8: Sistema Z-N en BC para roll

Se obtiene la misma respuesta que para la altura, la misma oscilación mantenida de TC = 6.283 segundos:

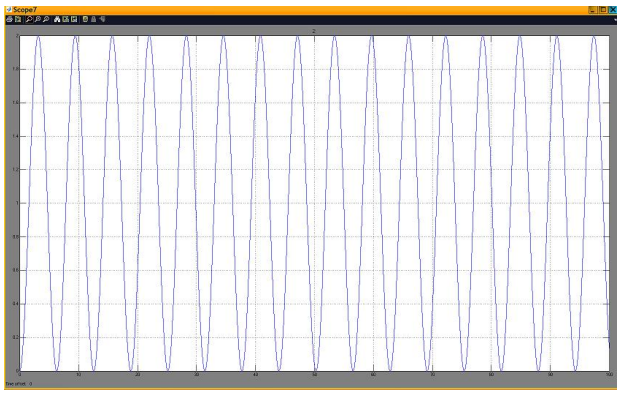


Figura C.9: Oscilaciones mantenidas en roll

Y por tanto el PID y la respuesta obtenida para el método de Z-N será el mismo:

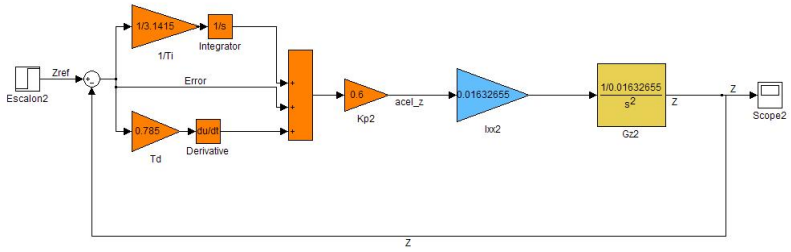


Figura C.10: Sistema controlado con PID de Z-N

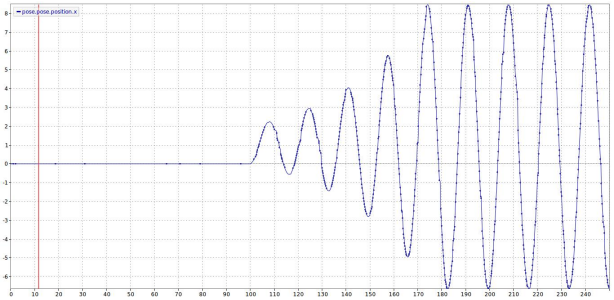
En Matlab, la respuesta en roll es equivalente a la que se obtuvo en Z, por lo que se llega al mismo PID. Sin embargo, y como se comprueba en durante el control en la sección dedicada para ello de la memoria, no serán iguales en el simulador tridimensional Gazebo, llegándose a unas sintonizaciones de los PID diferentes.

C.3 PID en X e Y

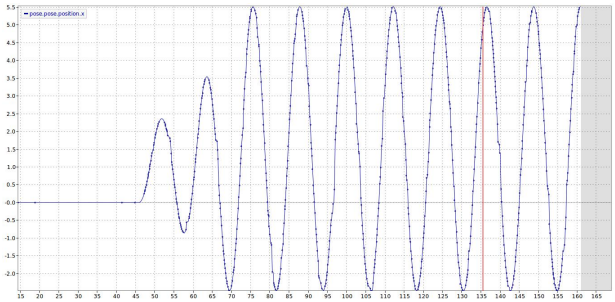
Como se comenta a lo largo de la memoria, X e Y se controlan mediante unos bucles de control externos sobre los controladores de *pitch* y de *roll* respectivamente, semejante a unos controladores Maestro-Esclavo, por lo que su sintonización se realiza directamente sobre el modelo en Gazebo utilizando también el método de Ziegler-Nichols en bucle cerrado.

Añadiendo el bucle externo de control sobre el pitch, de momento solo con acción proporcional de ganancia K_p, se busca ahora aumentando poco a poco la ganancia obtener una respuesta en forma de oscilaciones mantenidas. En la Figura C.11 se tienen 3 respuestas en las que se muestran las diferentes respuestas que se obtuvieron hasta llegar a la K_c final de valor 0.13.

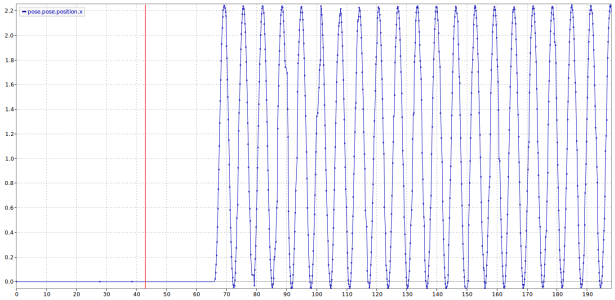
Para esta $K_c = K_p = 0{,}13$, con un periodo crítico de $T_C = 6{,}45$ segundos, el PID de Ziegler-Nichols sería:



(a) Kp = 0.015



(b) Kp = 0.03



(c) Kp = 0.13

Figura C.11: Método de Z-N para X e Y

$$\begin{cases} K_p = 0{,}078 \\ T_i = 3{,}225 \\ T_d = 0{,}80625 \end{cases} \tag{C.5}$$

Que en su forma en paralelo sería:

$$\begin{cases} K_p = 0{,}078 \\ T_i = 41{,}35 \\ T_d = 0{,}0629 \end{cases} \tag{C.6}$$

Añadiendo este PID se obtiene una respuesta con mucha sobreoscilación, por lo que se realiza una sintonización fina manualmente en la que

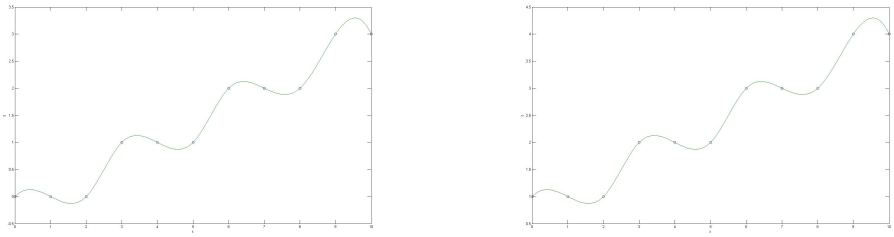
predominantemente se va aumentado el término derivativo y disminuyendo el integral (aumentando T_i) hasta llegar al PID con el que nos quedamos y que simulará en el apartado correspondiente:

$$\begin{cases} K_p = 0{,}12 \\ T_i = 80 \\ T_d = 0{,}18 \end{cases} \tag{C.7}$$

Apéndice D

Algoritmo Look-Ahead para el seguimiento de trayectorias

Para el seguimiento de una trayectoria, lo primero que hace falta es esta misma, bien establecida de antemano o bien vaya llegando de un sistema de navegación basado en sensores. En este caso, se trata de la primera opción y se ha generado en Matlab mediante la función de interpolación cúbica *spline()* dados unos puntos (X, Y, Z) de interés, representados por círculos en las siguientes gráficas:



(a) Trayectoria Y-X

(b) Trayectoria Z-X

Figura D.1: Trayectoria

Una vez tenemos la trayectoria, lo siguiente es establecer el algoritmo por el cual se van seleccionar los puntos a enviar al sistema de control de posición del quadrotor. El algoritmo programado en este proyecto se basa en seleccionar como próximo punto deseado a enviar (x_d, y_d, z_d), en el que posicionar el quadrotor, aquel perteneciente a la trayectoria 'hacia delante'que se encuentre a una distancia **L** (*Look-Ahead*) de la posición actual del quadrotor (x_a, y_a, z_a) en el plano XY. Con "hacia delante" se hace referencia de que el punto no debe ser uno por el que ya se haya pasado o simplemente se haya dejado atrás. Visto gráficamente, sería tal y como se muestra en la Figura D.2.

Para su programación hay tener en cuenta que la trayectoria está dada por puntos discretos, por lo que difícilmente va a existir un punto de la trayectoria exactamente a la distancia L de la posición actual, lo que hace que tengamos que ir comprobando punto a punto de la trayectoria cual es

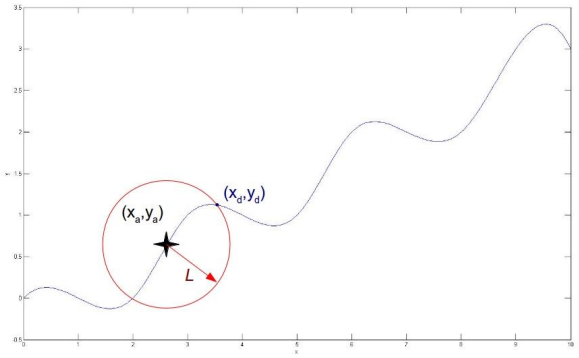


Figura D.2: Seguimiento Look-Ahead

el punto más lejano a una distancia menor que L.

El algoritmo se basará en los aspectos geométricos representados en la Figura D.2, de tal forma que en cada iteración se tendrán:

- Ultima posición enviada de la trayectoria: (x_k, y_k, z_k)
- Posición actual del quadrotor: (x_a, y_a, z_a)
- Radios equivalentes: $R_i = \sqrt{(x_i - x_a)^2 + (y_i - y_a)^2}$, con i desde k hasta n, siendo n el índice del último punto de la trayectoria, siendo (x_i, y_i, z_i) un punto perteneciente a la trayectoria.

De esta forma, para hallar el próximo punto (x_d, y_d, z_d) a enviar se define para cada punto de la trayectoria a partir del último que se envió, un radio equivalente R y se comprueba que sea menor que el *Look-Ahead*. El último punto de la trayectoria cuyo radio equivalente sea menor que el *Look-Ahead* será el punto a enviar, la próxima posición deseada.

El algoritmo constaría de los siguientes pasos:

1. Leer posición actual del quadrotor, (x_a, y_a, z_a).
2. Incrementar índice i.
3. Calcular el radio equivalente, R_i, del punto de la trayectoria (x_i, y_i, z_i).
4. Comprobar que $R_i \leq L$ y $i < n$. En caso positivo volver al punto 2.
5. Enviar posición deseada, $(x_d, y_d, z_d) = (x_{i-1}, y_{i-1}, z_{i-1})$, por servicio.
6. Actualizar último punto enviado, $(x_k, y_k, z_k) = (x_d, y_d, z_d)$, y el índice, $i = k$.
7. Si $i < n$ volver al punto 2. Si no, fin de la trayectoria.

Todo esto, queda programado en *gen_ trayectoria.cpp* y puede verse en el **apéndice B**, al igual que los puntos de la trayectoria de prueba en el archivo *trayectoria.h*.

Apéndice E

Plano Manipulador

A continuación se presentan, en un mismo plano, el brazo manipulador junto con sus dimensiones principales y una lista de piezas con datos relativos al material y el peso de los componentes principales.

La acotación de los elementos no ha pretendido ser completa, al no ser objeto del proyecto el diseño del manipulador, siendo obra de D. Jesús Martín Sánchez, y bastando para la elaboración del modelo con las dimensiones acotadas.

Apéndice F

Códigos Manipulador

F.1 manupulador.urdf

```
<?xml version="1.0"?>
<robot name="manipulador">

<!--********************ESLABONES*********************-->
<link name="Conjunto_base">
  <visual>
    <geometry>
     <mesh filename="package://manipulador/visual/Conjunto_base_completo.
         dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    </visual>
  <collision>
    <geometry>
     <mesh filename="package://manipulador/visual/Conjunto_base_completo.
         dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
     <origin xyz="3.936916e-3 -23.544251e-3 -40.343050e-3" rpy="0 0 0"/>
     <mass value="0.214623"/>
     <inertia ixx="3618.2982e-7" ixy="-549.4840e-7" ixz="-122.1871e-7"
              iyy="1151.3618e-7" iyz="470.4171e-7" izz="3685.7680e-7"/>
  </inertial>
</link>

<link name="Bateria">
  <visual>
   <geometry>
     <box size="0.043 0.150 0.018"/>
     </geometry>
      <material name="amarillo">
      <color rgba="1 1 0 1"/>
     </material>
  </visual>
  <collision>
    <geometry>
```

```
      <box size="0.043 0.150 0.018"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0.231"/>
    <inertia ixx="4416.44e-7" ixy="0.0" ixz="0.0"
             iyy="420.475e-7" iyz="0.0" izz="4711.531e-7"/>
  </inertial>
</link>
<gazebo reference="Bateria">
   <material>Gazebo/Yellow</material>
</gazebo>

<link name="Brazo_1">
  <visual>
    <geometry>
     <mesh filename="package://manipulador/visual/brazo1.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://manipulador/visual/brazo1.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="-5.6536e-3 0.393e-3 -102.056e-3" rpy="0 0 0"/>
    <mass value="0.101602"/>
    <inertia ixx="7595.151e-7" ixy="-2.2392e-7" ixz="312.682e-7"
             iyy="8157.2174e-7" iyz="-7.9513e-7" izz="776.2473e-7"/>
  </inertial>
</link>

<link name="Brazo_2">
  <visual>
    <geometry>
     <mesh filename="package://manipulador/visual/brazo2.dae"
          scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://manipulador/visual/brazo2.dae"
           scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="-1.883e-3 94.381e-3 1.436e-3" rpy="0 0 0"/>
    <mass value="0.036285"/>
    <inertia ixx="1041.690e-7" ixy="-31.590e-7" ixz="1.178e-7"
             iyy="143.044e-7" iyz="0.430e-7" izz="1126.690e-7"/>
```

```
  </inertial>
</link>

<link name="Chasis_efector">
  <visual>
    <geometry>
      <mesh filename="package://manipulador/visual/Chasis_efector.dae"
            scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://manipulador/visual/Chasis_efector.dae"
            scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="-4.5271e-3 8.1896e-3 -0.8998e-3" rpy="0 0 0"/>
    <mass value="0.016902"/>
    <inertia ixx="14.33064e-7" ixy="-0.364275e-7" ixz="-0.589219e-7"
             iyy="21.987118e-7" iyz="-0.146395e-7" izz="15.210762e-7"/>
  </inertial>
</link>

<link name="Gancho_efector">
  <visual>
    <geometry>
      <mesh filename="package://manipulador/visual/Gancho_efector.dae"
            scale="0.0255 0.0255 0.0255"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://manipulador/visual/Gancho_efector.dae"
            scale="0.0255 0.0255 0.0255"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0.809174e-3 8.634374e-3 -3.791276e-3" rpy="0 0 0"/>
    <mass value="0.001593"/>
    <inertia ixx="1.430595e-7" ixy="0.009745e-7" ixz="-0.008666e-7"
             iyy="3.17436e-7" iyz="-0.011176e-7" izz="3.715269e-7"/>
  </inertial>
</link>

<link name="fantasma">
  <visual>
    <geometry>
      <box size="0.1 0.01 0.01"/>
    </geometry>
    <material name="Azulito">
      <color rgba="0 0 0.5 0.5"/>
    </material>
```

```
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0.0"/>
    <inertia ixx="0.0" ixy="0.0" ixz="0.0"
             iyy="0.0" iyz="0.0" izz="0.0"/>
  </inertial>
</link>

<!--******************ARTICULACIONES******************-->
<joint name="Bandeja_bateria" type="prismatic">
   <parent link="Conjunto_base"/>
   <child link="Bateria"/>
   <origin rpy="0 0 0" xyz="0 -30e-3 -11e-3"/>
   <axis xyz="0 1 0"/>
   <limit lower="-0.04" upper="0" velocity="6.98" effort="0.94"/>
   <safety_controller k_velocity="2.04" k_position="10" soft_lower_limit=
       "-1" soft_upper_limit="1.5"/>
</joint>

<joint name="Articulacion_1" type="revolute">
   <parent link="Conjunto_base"/>
   <child link="Brazo_1"/>
   <origin rpy="0 0 0" xyz="0 26.00e-3 -55.00e-3"/>
   <axis xyz="1 0 0"/>
   <limit lower="-1.08" upper="1.57" velocity="6.98" effort="0.94"/>
   <safety_controller k_velocity="2.04" k_position="10" soft_lower_limit=
       "-1" soft_upper_limit="1.5"/>
</joint>

<joint name="Articulacion_2" type="revolute">
   <parent link="Brazo_1"/>
   <child link="Brazo_2"/>
   <origin rpy="0 0 0" xyz="0 0 -200e-3"/>
   <axis xyz="1 0 0"/>
   <limit lower="-2.8916" upper="0.25" velocity="6.16" effort="0.64"/>
   <safety_controller k_velocity="2.04" k_position="10" soft_lower_limit=
       "-2.7" soft_upper_limit="0.25" />
</joint>

<joint name="Articulacion_3" type="revolute">
   <parent link="Brazo_2"/>
   <child link="Chasis_efector"/>
   <origin rpy="0 0 0" xyz="0 155.77e-3 0"/>
   <axis xyz="0 1 0"/>
   <limit lower="-1.047" upper="1.047" velocity="9.52" effort="0.22"/>
   <safety_controller k_velocity="0.145" k_position="100"
       soft_lower_limit="-0.9" soft_upper_limit="0.9"/>
</joint>

<joint name="Articulacion_efector" type="revolute">
   <parent link="Chasis_efector"/>
   <child link="Gancho_efector"/>
   <origin rpy="0 0 0" xyz="-4.5e-3 9.5e-3 -6.5e-3"/>
```

```
   <axis xyz="1 0 0"/>
   <limit lower="-1" upper="0.6" velocity="9.52" effort="0.22"/>
   <!--
   <safety_controller k_velocity="0.42" k_position="19.04"
       soft_lower_limit="-1"
   soft_upper_limit="0.6"/>-->
   <!--La apertura y cierre del gancho sin Safety control, el software no
        esta preparado para esto
   -->
</joint>

<joint name="Articulacion_fantasma" type="continuous">
   <parent link="Brazo_2"/>
   <child link="fantasma"/>
   <origin rpy="0 0 0" xyz="0 178e-3 -9e-3"/>
   <axis xyz="1 0 0"/>
</joint>

<!--******************TRANSMISIONES******************-->
<transmission name="Transmision_1" type="pr2_mechanism_model/
    SimpleTransmission">
   <actuator name="Motor_1" />
   <joint name="Articulacion_1" />
   <mechanicalReduction>1</mechanicalReduction>
</transmission>

<transmission name="Transmision_2" type="pr2_mechanism_model/
    SimpleTransmission">
   <actuator name="Motor_2" />
   <joint name="Articulacion_2" />
   <mechanicalReduction>1</mechanicalReduction>
</transmission>

<transmission name="Transmision_3" type="pr2_mechanism_model/
    SimpleTransmission">
   <actuator name="Motor_3" />
   <joint name="Articulacion_3" />
   <mechanicalReduction>1</mechanicalReduction>
</transmission>

<transmission name="Transmision_efector" type="pr2_mechanism_model/
    SimpleTransmission">
   <actuator name="Motor_efector" />
   <joint name="Articulacion_efector" />
   <mechanicalReduction>1</mechanicalReduction>
</transmission>

<transmission name="Transmision_fantasma" type="pr2_mechanism_model/
    SimpleTransmission">
   <actuator name="Motor_fantasma" />
   <joint name="Articulacion_fantasma" />
   <mechanicalReduction>1</mechanicalReduction>
</transmission>

</robot>
```

F.2 control_cartesiano_manipulador.h

```
// Creacion de una clase propia de controlador, dentro un namespace
    propio controlador_ns, heredera de la clase base de los controladores
     de pr2_controller_interface::Controller
// Se sobreescriben algunos metodos cuya definicion se encuentran en
    controlador.cpp

// Include funciones comunes std y ROS
#include <ros/ros.h>
// Incluimos la clase Controller a heredar
#include <pr2_controller_interface/controller.h> // Includes
necesarios para usar controlador
// Incluimos las clases JointState y RobotState
#include <pr2_mechanism_model/joint.h> #include
<pr2_mechanism_model/robot.h>
// Incluimos tipo de mensaje de los servicios de cambio de posicion
    deseada
#include <control_cartesiano_manipulador/Cambio_Consigna.h>
// Incluimos la clase Pid
#include <control_toolbox/pid.h>
// Incluimos el servicio vacio basico de ROS
#include <std_srvs/Empty.h>
// Incluimos el tipo de mensaje para obtener informacion del controlador
    via topic
#include <control_cartesiano_manipulador/Lectura_datos.h>
// Incluimos clase cadena de pr2_mechanism
#include <pr2_mechanism_model/chain.h>
// Incluimos clase transmision de pr2_mechanism
#include <pr2_mechanism_model/simple_transmission.h>
// Incluimos tipo de mensaje
#include <control_cartesiano_manipulador/Pos_des_efector.h>
// Incluimos las clases necesarias de KDL
// (Cadenas, Vectores de articulaciones, solvers cinematicos,...)
#include <boost/scoped_ptr.hpp>
#include <kdl/chain.hpp>
#include <kdl/chainfksolverpos_recursive.hpp>
#include <kdl/chainiksolverpos_nr.hpp>
#include <kdl/chainiksolver.hpp>
#include <kdl/chainiksolverpos_nr_jl.hpp>
#include <kdl/chainiksolvervel_pinv.hpp> #include <kdl/frames.hpp>
#include <kdl/jntarray.hpp>

namespace control_cartesiano_nspace{

// Nuestra ClaseControlCartesiano debe heredar obligatoriamente la forma
    de la clase Controller.
class ClaseControlCartesiano: public pr2_controller_interface::Controller
     {

 private:
  // Objeto a la clase RobotState, para guardar el que le llega a control
      .cpp en init()
  pr2_mechanism_model::RobotState *manipulador;
```

```
// Puntero a los objetos JointState. Uno por articulacion para acceder
    a cada una
pr2_mechanism_model::JointState* articulacion_state_1;
pr2_mechanism_model::JointState* articulacion_state_2;
pr2_mechanism_model::JointState* articulacion_state_3;
pr2_mechanism_model::JointState* articulacion_state_efector;

// Objeto del tipo cadena de pr2_mechanis_model, que contiene el objeto
     KDL, entre otras cosas, que es la que nos interesa
pr2_mechanism_model::Chain cadena;

// Objeto del tipo de KDL, necesario tener la cadena en este tipo para
    usar las funciones solvers KDL
KDL::Chain cadena_kdl;

// Posiciones articulares actuales
KDL::JntArray q; //[q1, ... , qn]. El tamano se impondra al leer la
    cadena y saber el numero de joints
// Posiciones articulares iniciales
KDL::JntArray q0;
// Posiciones articulares deseadas
KDL::JntArray qd;

// Posicion efector actual
KDL::Frame x; //x=[p,M]=[vector de posicion 3x1, matriz de rotacion 3x3
    ]
// Posicion efector deseada
KDL::Frame xd;
// Posicion efector inicial
KDL::Frame x0;

// KDL Solvers: Se reserva memoria dinamicamente para este tipo de
    objetos
boost::scoped_ptr<KDL::ChainFkSolverPos> solver_CD;
boost::scoped_ptr<KDL::ChainIkSolverPos> solver_CI;
// Equivalente pero en velocidad, necesario como argumento del
    solver_CI
boost::scoped_ptr<KDL::ChainIkSolverVel_pinv> solver_CI_Vel;

// Objeto Time para guardar el instante en que empezo el ciclo de
    control actual
ros::Time fin_ciclo;

// Servicio modificacion consigna del efector (abrir/cerrar pinza):
double consigna_efector;
ros::ServiceServer srv_gancho_efector;
bool cambio_gancho_efector(control_cartesiano_manipulador::
    Cambio_Consigna::Request& req, control_cartesiano_manipulador::
    Cambio_Consigna::Response& resp);

// Servicios posicion deseada en (x),y,z y pitch.
double pitch_deseado_efector;
ros::ServiceServer srv_pos_efector;
ros::ServiceServer srv_pitch_efector;
bool cambio_pos_efector(control_cartesiano_manipulador::Pos_des_efector
```

```
      ::Request& req, control_cartesiano_manipulador::Pos_des_efector::
      Response& resp);
  bool cambio_pitch_efector(control_cartesiano_manipulador::
      Cambio_Consigna::Request& req, control_cartesiano_manipulador::
      Cambio_Consigna::Response& resp);

  // PID: Objetos del tipo Pid
  control_toolbox::Pid pid_1;
  control_toolbox::Pid pid_2;
  control_toolbox::Pid pid_3;
  control_toolbox::Pid pid_efector;

  // Lectura de datos (TOPIC)
    // Art1
     ros::Publisher topic_pub_1;
     control_cartesiano_manipulador::Lectura_datos Datos_1;
    // Art 2
     ros::Publisher topic_pub_2;
     control_cartesiano_manipulador::Lectura_datos Datos_2;
    // Art 3
     ros::Publisher topic_pub_3;
     control_cartesiano_manipulador::Lectura_datos Datos_3;
    // Efector
     ros::Publisher topic_pub_efector;
     control_cartesiano_manipulador::Lectura_datos Datos_efector;

 public:
  // Funciones miembro del controlador (def en .cpp)
  virtual bool init(pr2_mechanism_model::RobotState *robot,
                    ros::NodeHandle &n);
  virtual void starting();
  virtual void update();
  virtual void stopping();
}; }
```

F.3 control_cartesiano_manipulador.cpp

```
// Definiciones de los metodos sobreescribidos en nuestra clase
    controlador
// Incluimos el archivo de cabecera con la declaracion de la clase
    ClaseControl
#include <control_cartesiano_manipulador/control_cartesiano_manipulador.h
    >
#include <pluginlib/class_list_macros.h>

using namespace control_cartesiano_nspace;

/// Inicializacion del controlador en tiempo NO real
bool ClaseControlCartesiano::init(pr2_mechanism_model::RobotState *robot,
     ros::NodeHandle &n)
  {
    // Se recibe en *robot el objeto RobotState del manipulador y el
        nodehandle que pr2_manager haya asignado a este proceso para
```

```
    poder acceder desde esta funcion al Serv de Param

// Obtenemos del Serv de Param el nombre de las articulaciones a
    controlar definidas en el archivo yaml y las guardamos como
    cadenas de caract
  std::string joint_name_1;
  if (!n.getParam("joint_name_1", joint_name_1))
  {
    ROS_ERROR("Error al buscar las articulaciones: No joint given in
        namespace: '%s')", n.getNamespace().c_str());
    return false;
  }
  std::string joint_name_2;
  if (!n.getParam("joint_name_2", joint_name_2))
  {
    ROS_ERROR("Error al buscar las articulaciones: No joint given in
        namespace: '%s')", n.getNamespace().c_str());
    return false;
  }
  std::string joint_name_3;
  if (!n.getParam("joint_name_3", joint_name_3))
  {
    ROS_ERROR("Error al buscar las articulaciones: No joint given in
        namespace: '%s')", n.getNamespace().c_str());
    return false;
  }
  std::string joint_name_efector;
  if (!n.getParam("joint_name_efector", joint_name_efector))
  {
    ROS_ERROR("Error al buscar las articulaciones: No joint given in
        namespace: '%s')", n.getNamespace().c_str());
    return false;
  }

// Obtenemos los JointState de cada articulacion a partir del
    RobotState por su nombre
  articulacion_state_1 = robot->getJointState(joint_name_1);
  if (!articulacion_state_1)
  {
    ROS_ERROR("Error al buscar la articulacion '%s'", joint_name_1.
        c_str());
    return false;
  }
  articulacion_state_2 = robot->getJointState(joint_name_2);
  if (!articulacion_state_2)
  {
    ROS_ERROR("Error al buscar la articulacion '%s'", joint_name_2.
        c_str());
    return false;
  }
  articulacion_state_3 = robot->getJointState(joint_name_3);
  if (!articulacion_state_3)
  {
    ROS_ERROR("Error al buscar la articulacion '%s'", joint_name_3.
        c_str());
```

```
    return false;
  }
  articulacion_state_efector = robot->getJointState(
      joint_name_efector);
  if (!articulacion_state_efector)
  {
    ROS_ERROR("Error al buscar la articulacion '%s'",
        joint_name_efector.c_str());
    return false;
  }

// Obtenemos del Serv de Param el nombre de los links inicial (base o
    root) y final (pre-efector o tip) que formaran la cadena,
    definidos y cargado en el yaml
  std::string root_name, tip_name;
  if (!n.getParam("root_name", root_name))
  {
      ROS_ERROR("No existe el parametro root_name en %s)",
      n.getNamespace().c_str());
      return false;
  }
  if (!n.getParam("tip_name", tip_name))
  {
      ROS_ERROR("No existe el parametro tip_name en %s)",
      n.getNamespace().c_str());
      return false;
  }

 // Copiamos el objeto robot al declarado en controlador.h para no
     perderlo fuera de esta funcion
     manipulador = robot;
 // Construimos el objeto cadena de la clase pr2::mechanism::chain del
      cual obtendremos la KDL::Chain
  if (!cadena.init(robot, root_name, tip_name))
  {
      ROS_ERROR("MyCartController could not use the chain from '%s'
          to '%s'",
      root_name.c_str(), tip_name.c_str());
      return false;
  }

 // obtenemos la cadena kdl a partir de la pr2_mechanism
  cadena.toKDL(cadena_kdl);
 // Damos el tamano apropiado a nuestras variabes una vez que ya
     sabemos el numero de joints de la cadena
  q.resize(cadena_kdl.getNrOfJoints());
  q0.resize(cadena_kdl.getNrOfJoints());
  qd.resize(cadena_kdl.getNrOfJoints());
 // Construimos los solvers de la CD en posicion y de la CI en
     velocidad para la cadena anterior (kdl)
  solver_CD.reset(new KDL::ChainFkSolverPos_recursive(cadena_kdl));
  solver_CI_Vel.reset(new KDL::ChainIkSolverVel_pinv(cadena_kdl));
 // Construimos el solver de la CI (con metodo de Newton-Raphson) en
     posicion para la cadena (kdl),
 // pasandole como argumento los solvers necesarios para el metodo de
```

```
    iteracion interno
  solver_CI.reset(new KDL::ChainIkSolverPos_NR(cadena_kdl,*solver_CD
      ,*solver_CI_Vel));


// Servicios para la apertura y cierre del efector
    srv_gancho_efector = n.advertiseService("cambio_gancho_efector",
        &ClaseControlCartesiano::cambio_gancho_efector, this);
// Servicios para cambiar la posicion (X,Y,Z) deseada del TCP. X
    siempre cero por la configuracion del manipulador
    srv_pos_efector = n.advertiseService("cambio_pos_efector", &
        ClaseControlCartesiano::cambio_pos_efector, this);
// Servicios para cambiar la orientacion (pitch) deseada del TCP.
    srv_pitch_efector = n.advertiseService("cambio_pitch_efector", &
        ClaseControlCartesiano::cambio_pitch_efector, this);

// Variable con la posicion articular deseada para el efector a
    modificar en el servicio de apertura y cierre del efector
// Inicialmente a 0 que equivale a la posicion de equilibrio (ni
    abierta ni cerrada)
    consigna_efector = 0;
// Variable del pitch deseado que entra por servicio (angulo de la
    tercera articulacion)
    pitch_deseado_efector = 0;

// Construimos los PIDs con los valores de los coeficientes definidos
    en el Serv de Param
    if (!pid_1.init(ros::NodeHandle(n, "pid_parameters_1")))
    {
        ROS_ERROR("Fallo en la construccion del PID para la '%s'",
            joint_name_1.c_str());
        return false;
    }
    if (!pid_2.init(ros::NodeHandle(n, "pid_parameters_2")))
    {
        ROS_ERROR("Fallo en la construccion del PID para la '%s'",
            joint_name_2.c_str());
        return false;
    }
    if (!pid_3.init(ros::NodeHandle(n, "pid_parameters_3")))
    {
        ROS_ERROR("Fallo en la construccion del PID para la '%s'",
            joint_name_3.c_str());
        return false;
    }
    if (!pid_efector.init(ros::NodeHandle(n, "pid_parameters_efector")
       ))
    {
        ROS_ERROR("Fallo en la construccion del PID para la '%s'",
            joint_name_efector.c_str());
        return false;
    }

 // Topics para envio de datos
    topic_pub_1 = n.advertise<control_cartesiano_manipulador::
```

```
            Lectura_datos>("Lectura_datos_articulacion_1", 10000);
        topic_pub_2 = n.advertise<control_cartesiano_manipulador::
            Lectura_datos>("Lectura_datos_articulacion_2", 10000);
        topic_pub_3 = n.advertise<control_cartesiano_manipulador::
            Lectura_datos>("Lectura_datos_articulacion_3", 10000);
        topic_pub_efector = n.advertise<control_cartesiano_manipulador::
            Lectura_datos>("Lectura_datos_articulacion_efector", 10000);

   return true;
}

/// Inicio del controlador en tiempo real (se ejecuta solo 1 vez)
void ClaseControlCartesiano::starting() {
    // Posiciones articulares iniciales (Lectura encoders)
       cadena.getPositions(q0);
    // Posiciones cartesianas iniciales
       solver_CD->JntToCart(q0, x0);
    // Posicion deseada inicial igual a la inicial
       xd=x0;
     // Obtenemos el instante actual (NO ES EL MISMO QUE ros::Time::now()
         .
     // Pr2_controller_manager utiliza este tiempo de los RobotState para
          lanzar todos los controladores a la vez)
       fin_ciclo = manipulador->getTime();
     // Reseteamos los PIDs al inicio
      pid_1.reset();
      pid_2.reset();
      pid_3.reset();
      pid_efector.reset();
}

/// Actualizacion en cada ciclo del controlador (se ejecuta en tiempo
    real a 1000Hz)
void ClaseControlCartesiano::update() {
    // Posicion actual articulaciones (lectura encoders)
    cadena.getPositions(q);
    // Comprobacion
    //~ ROS_INFO("Posicion articulaciones: [q1,q2,q3]=[ %f , %f , %f ]",
         q(0), q(1), q(2));

    // Posicion actual TCP (modelo CD)
    solver_CD->JntToCart(q, x);
    // COMPROBACIONES
    //~ ROS_INFO("Posicion actual del efector: [x,y,z]=[ %f , %f , %f
        ]", x.p.x(), x.p.y(), x.p.z()); // x.p(0)=x.p.x()

    // Posiciones deseada TCP (modelo CI) --> Obtenemos qd= (q1,q2,
        qfantasma) a partir de xd (que se modifica por servicio)
    int ok=solver_CI->CartToJnt(q, xd, qd);
     if (ok<0)
      {
        ROS_INFO("FALLO EN LA CINEMATICA INVERSA (se llego al numero
            maximo de iteraciones o no se consiguio el minimo deseado en
            el error en el metodo Newton R.)");
      }
```

```
 else
  {
    ROS_INFO("Posicion articulaciones deseada: [q1,q2]=[ %f , %f]",
        qd(0), qd(1));
  }

// CONTROL PID ARTICULACIONES a parti de los valores deseados pasados
     a las articulaciones
double pos_des_1 = qd(0) ; // Posicion deseada articulacion 1
double pos_act_1 = q(0) ; // Posicion actual articulacion 1
double pos_des_2 = qd(1) ;
double pos_act_2 = q(1) ;
double pos_des_3 = pitch_deseado_efector ; // INDEPENDIENTE DE LA
    RESOLUCION DE LA CINEMATICA INVERSA (SE CAMBIO POR EL LINK
    FANTASMA)
double pos_act_3 = articulacion_state_3->position_;
double pos_des_efector = consigna_efector ;
double pos_act_efector = articulacion_state_efector->position_;
// Incremento de tiempo entre la ultima accion de control y esta. Se
    le pasa al PID para el calculo de la integral y la derivada
ros::Duration dt = manipulador->getTime() - fin_ciclo;
// Se actualiza el "instante de aplicacion de la ultima accion de
    control", que pasa a ser ahora
fin_ciclo = manipulador->getTime();
// Se le aplica un esfuerzo a cada articulacion, por medio del
    miembro commanded_effort de nuestro objeto JointState,
// igual al calculado por nuestro PID para un error y un incremento
    de tiempo dados
articulacion_state_1->commanded_effort_ = pid_1.updatePid(pos_act_1-
    pos_des_1, dt);
articulacion_state_2->commanded_effort_ = pid_2.updatePid(pos_act_2-
    pos_des_2, dt);
articulacion_state_3->commanded_effort_ = pid_3.updatePid(pos_act_3-
    pos_des_3, dt);
// En el caso del cierre del efector se corta el esfuerzo en 0.005 Nm
     para evitar inestabilidades en la sujecion de los objetos
if(consigna_efector!=0.45)
{
    articulacion_state_efector->commanded_effort_ = pid_efector.
        updatePid(pos_act_efector-pos_des_efector, dt);
}
else
{
    articulacion_state_efector->commanded_effort_ = 0.005;
}

// Rellenamos informacion y enviamos por topic
Datos_1.tiempo = ros::Time::now().toSec();
Datos_1.dt = dt.toSec();
Datos_1.position = articulacion_state_1->position_;
Datos_1.desired_position = pos_des_1;
Datos_1.velocity = articulacion_state_1->velocity_;
Datos_1.commanded_effort = articulacion_state_1->commanded_effort_;
Datos_1.measured_effort = articulacion_state_1->measured_effort_;
topic_pub_1.publish(Datos_1);
```

```
    Datos_2.tiempo = ros::Time::now().toSec();
    Datos_2.dt = dt.toSec();
    Datos_2.position = articulacion_state_2->position_;
    Datos_2.desired_position = pos_des_2;
    Datos_2.velocity = articulacion_state_2->velocity_;
    Datos_2.commanded_effort = articulacion_state_2->commanded_effort_;
    Datos_2.measured_effort = articulacion_state_2->measured_effort_;
    topic_pub_2.publish(Datos_2);
    Datos_3.tiempo = ros::Time::now().toSec();
    Datos_3.dt = dt.toSec();
    Datos_3.position = articulacion_state_3->position_;
    Datos_3.desired_position = pos_des_3;
    Datos_3.velocity = articulacion_state_3->velocity_;
    Datos_3.commanded_effort = articulacion_state_3->commanded_effort_;
    Datos_3.measured_effort = articulacion_state_3->measured_effort_;
    topic_pub_3.publish(Datos_3);
    Datos_efector.tiempo = ros::Time::now().toSec();
    Datos_efector.dt = dt.toSec();
    Datos_efector.position = articulacion_state_efector->position_;
    Datos_efector.desired_position = pos_des_efector;
    Datos_efector.velocity = articulacion_state_efector->velocity_;
    Datos_efector.commanded_effort = articulacion_state_efector->
        commanded_effort_;
    Datos_efector.measured_effort = articulacion_state_efector->
        measured_effort_;
    topic_pub_efector.publish(Datos_efector);
}

/// Parada del controlador (in realtime)
void ClaseControlCartesiano::stopping() {}


/// SERVICIO CAMBIO CONSIGNA EFECTOR -- Funcion que el servicio
    srv_consigna_efector lanzara (paralelamente al update() ojo, ya que
    es en tiempo real) al ser llamado desde la terminal o desde otro
    programa
bool
ClaseControlCartesiano::cambio_gancho_efector(
    control_cartesiano_manipulador::Cambio_Consigna::Request&
req, control_cartesiano_manipulador::Cambio_Consigna::Response&
resp) {
     if (req.consigna==1) // Cerrar pinza
       {
           consigna_efector= 0.45; // La posicion maxima de cierre de la
               articulacion del efector (archivo URDF)
           resp.confirmacion = true;
           ROS_INFO("Controlador cerrando la pinza");
       }
     if (req.consigna==0) // Posicion de equilibrio
       {
           consigna_efector= 0;
           resp.confirmacion = true;
           ROS_INFO("Controlador devolviendo la pinza a su posicion de
               equilibrio");
       }
```

```
      if (req.consigna==-1) // Abrir pinza
        {
            consigna_efector= -0.8; // La posicion de maxima de apertura
                de la articulacion del efector
            resp.confirmacion = true;
            ROS_INFO("Controlador abriendo la pinza");
        }
      if (req.consigna!=1&&req.consigna!=0&&req.consigna!=-1)
       {
        ROS_ERROR("No se eligio bien la opcion de abrir/cerrar la pinza.
            Seleccione valores 1,0 o -1");
        resp.confirmacion = false; // Para indicar que se ha elegido mal
            al programa que llamo al servicio
        return false;
       }
      return true;
}


/// SERVICIOS CAMBIO POSICION DESEADA DEL EFECTOR
bool

ClaseControlCartesiano::cambio_pos_efector(control_cartesiano_manipulador
    ::Pos_des_efector::Request& req,
control_cartesiano_manipulador::Pos_des_efector::Response& resp) {
    // Modo de empleo desde terminal:
    //$ rosservice call /controlador_cartesiano/cambio_pos_efector '{
        posicion: {x: 0.0, y: 0.204, z: -0.264}}' que seria la posicion
        inicial

    // Trasladamos las posiciones deseadas respecto a la base al sistema
        de referencia del brazo 1 donde esta definido espacio de trabajo
        para ver si esta dentro
    float y_des_ref_1= req.posicion.y - 0.026;
    float z_des_ref_1= req.posicion.z + 0.055;

    if((sqrt((y_des_ref_1*y_des_ref_1)+(z_des_ref_1*z_des_ref_1))
        <=0.3780)&&(sqrt((y_des_ref_1*y_des_ref_1)+(z_des_ref_1*
        z_des_ref_1))>=0.2745)) // Si esta dentro del espacio de trabajo
        {
            xd.p(1)=req.posicion.y; // Asigno el valor que le llegue a la
                y_deseada ( ojo: respecto al sistema de referencia de la
                base que es en el que esta xd)
            xd.p(2)=req.posicion.z; // Asigno el valor que le llegue a la
                z_deseada
            ROS_INFO("Posicion deseada: Y: %f Z: %f", xd.p(1), xd.p(2));
            resp.confirmacion=true; // Y devolvemos en el servicio el
                logico true de que se asignado correctamente
        }
    else // Si no esta dentro del espacio de trabajo
        {
            resp.confirmacion=false; // Para indicar que se ha elegido mal
                 al programa que llamo al servicio
            ROS_ERROR("Posicion deseada del efector fuera del espacio de
                trabajo actual");
```

```
            return false;
        }
    return true;
}

/// SERVICIO CAMBIO DE PITCH
bool
ClaseControlCartesiano::cambio_pitch_efector(
    control_cartesiano_manipulador::Cambio_Consigna::Request&
req, control_cartesiano_manipulador::Cambio_Consigna::Response&
resp) {
    if((req.consigna<=1.047)&&(req.consigna>=-1.047)) // Limites del
        pitch
    {
            pitch_deseado_efector=req.consigna;
            resp.confirmacion=true;
    }
    else
    {
        resp.confirmacion=false; // Para indicar que se ha elegido mal al
             programa que llamo al servicio
        ROS_ERROR("Pitch deseado del efector fuera de rango");
        return false;
    }
    return true;
}


// Registramos el controlador en pluginlib para que el
    pr2_controller_manager tenga acceso
PLUGINLIB_DECLARE_CLASS(control_cartesiano_manipulador,
                     ControlCartesianoManipuladorPlugin,
                     control_cartesiano_nspace::ClaseControlCartesiano,
                     pr2_controller_interface::Controller)
```

F.4 planificación_manipulador.cpp

```
#include <ros/ros.h>
// Tipo de mensaje para el cliente al servicio para enviar posiciones
    deseadas al controlador del manipulador
#include <control_cartesiano_manipulador/Pos_des_efector.h>
// Tipo de mensaje para el cliente al servicio para enviar cambios de
    consigna a la articulacion 3 y a la pinza al controlador del
    manipulador
#include <control_cartesiano_manipulador/Cambio_Consigna.h>
// Tipo de mensaje propio para el servicio de objetivos
#include <manipulador/objetivo.h>
// Tipo de mensaje de Odometria (common_msg)
#include "nav_msgs/Odometry.h"
// Tipo de mensaje para representacion de posicion (x,y,z)
#include "geometry_msgs/Point.h"
// Para la lectura de ficheros URDF
#include <iostream> #include <fstream>
// Se incluye para tener acceso a la funcion de transformacion de
```

```
    orientaciones en RPY a Quaternion. Para SpawnModel la orientacion
    debe ser en Quaternion
#include "tf/transform_datatypes.h"
// Tipo de mensaje para representar orientacion en Quaternios
#include "geometry_msgs/Quaternion.h"
// Tipo de mensaje para lanzar cuerpos en Gazebo
#include "gazebo_msgs/SpawnModel.h"
// Tipo de mensaje para elminacion de esfuerzos en Gazebo
#include "gazebo_msgs/BodyRequest.h"
// Tipo de mensaje para elminacion modelos de Gazebo
#include "gazebo_msgs/DeleteModel.h"
// Include servicio vacio basico de ROS
#include <std_srvs/Empty.h>

// Cliente al servicio Pos des efector (del nodo de control cartesiano) y
     objeto del mismo tipo para enviar:
ros::ServiceClient client1;
control_cartesiano_manipulador::Pos_des_efector srv1;
// Cliente al servicio apertura/cierre efector (del nodo de control
    cartesiano) y objeto del mismo tipo para enviar:
ros::ServiceClient client2;
control_cartesiano_manipulador::Cambio_Consigna srv2;
// Cliente al servicio cambio pitch del efector (del nodo de control
    cartesiano) y objeto del mismo tipo para enviar:
ros::ServiceClient client3;
control_cartesiano_manipulador::Cambio_Consigna srv3;
// Cliente al servicio para enviar robots (barras a la posicion objetivo)
    a Gazebo (del nodo de Gazebo).
ros::ServiceClient client_spawn; gazebo_msgs::SpawnModel barra;
// Cliente al servicio para eliminar un cuerpo en Gazebo
ros::ServiceClient client_elimina_barra; gazebo_msgs::DeleteModel
elimina_barra;

// Posicion actual (link fantasma) refirida al sistema de referencia de
    la BASE (leido por Topic /sensor_pos_efector directamente de Gazebo)
geometry_msgs::Point pos_sensor; // Point=(x,y,z)
// Posicion deseada (link fantasma) refirida al sistema de referencia de
    la BASE, orientacion respecto al sistema de referencia del propio
    efector y tipo de objetivo (leido por Servicio /servicio_objetivo)
geometry_msgs::Point pos_obj; float orientacion; std::string
tipo_objetivo;

// Funcion auxiliar: Comprueba si el punto (definido respecto al Sref de
    la base del manipulador) que le llega como argumento esta dentro del
    espacio de trabajo de las articulaciones 1 y 2 definido respecto la
    articulacion 1
bool comprueba_espacio_de_trabajo (geometry_msgs::Point __pos_des)
    {
    /// Transformacion Sref Base a Sref Brazo_1, donde tenemos definido
        el espacio de trabajo
    float y_des_ref_1= __pos_des.y - 0.026;
    float z_des_ref_1= __pos_des.z + 0.055;

    if((sqrt((y_des_ref_1*y_des_ref_1)+(z_des_ref_1*z_des_ref_1))
        <=0.3780)&&(sqrt((y_des_ref_1*y_des_ref_1)+(z_des_ref_1*
```

```
        z_des_ref_1))>=0.2745)) // Si esta dentro del espacio de trabajo
        {return true;}
    else /// Si no esta dentro del espacio de trabajo
        {return false;}
    }

// Funcion auxiliar: Comprueba si la orientacion que le llega como
    argumento esta entre los limites de la articulacion 3
bool comprueba_orientacion (float __orientacion)
    {
    if((__orientacion<=1.047)&&(__orientacion>=-1.047)) // Comprobacion
        orientacion deseada dentro de Limites del pitch en la
        articulacion_3
        {return true;}
    else
        {return false;}
    }

// Funcion auxiliar: Calcula la posicion de aproximacion orientada
geometry_msgs::Point calcula_punto_orientado(float R,
geometry_msgs::Point pos_obj)
    {
    /// Transformacion del punto objetivo desde el Sref Base al Sref
        Brazo_1, donde tenemos definida esta funcion
    float a = pos_obj.y - 0.026;
    float b = pos_obj.z + 0.055;
    /// Longitudes de los brazos 1 y 2 mas el offset del link fantasma
    float L1 = 0.2 + 0.009;
    float L2 = 0.15577 + 0.02223;
    /// Variables que se utilizan mucho al cuadrado: pow(base,exponente)
    float a_2 = pow(a,2);
    float b_2 = pow(b,2);
    float R_2 = pow(R,2);
    float L1_2 = pow(L1,2);
    float L2_2 = pow(L2,2);
    /// Argumentos de los atan2
    float x1, x2, y1, y2;
    if(pos_obj.y>0) ///Primera solucion de q1 y q2 para el punto de
        aproximacion orientado. Valida en el 4 cuadrante, es decir, para
        y_objetivo > 0
        {
        ///y1,x1: argumentos del atan2() para hallar q1
        y1 = (b_2*a_2+b*sqrt(-a_2*(-R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*
            L2-L1_2)*(-R_2+a_2-2*L2*R-L2_2+b_2-2*L1*R-2*L1*L2-L1_2))+a_2*
            L1_2-a_2*L2_2-2*a_2*L2*R-a_2*R_2+pow(a,4))/(b_2+a_2)/L1/a/2.0;
        x1 = -(b*L1_2-b*L2_2+pow(b,3)-2*b*L2*R-b*R_2+b*a_2-sqrt(-a_2*(-
            R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*L2-L1_2)*(-R_2+a_2-2*L2*R-
            L2_2+b_2-2*L1*R-2*L1*L2-L1_2)))/(b_2+a_2)/L1/2.0;
        ///y2,x2: argumentos del atan2() para hallar q2
        y2 = (a_2+b_2-2*L2*R-L2_2-R_2-L1_2)/L1/(L2+R);
        x2 = sqrt(-a_2*(-R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*L2-L1_2)*(-
            R_2+a_2-2*L2*R-L2_2+b_2-2*L1*R-2*L1*L2-L1_2))/a/L1/(L2+R);
        }
    else ///Segunda solucion, valida en el 3 cuadrante, es decir, para
        y_objetivo < 0
```

```
        {
        ///y1,x1: argumentos del atan2() para hallar q1
        y1 = (b_2*a_2-b*sqrt(-a_2*(-R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*
            L2-L1_2)*(-R_2+a_2-2*L2*R-L2_2+b_2-2*L1*R-2*L1*L2-L1_2))+a_2*
            L1_2-a_2*L2_2-2*a_2*L2*R-a_2*R_2+pow(a,4))/(b_2+a_2)/L1/a/2.0;
        x1 = -(b*L1_2-b*L2_2+pow(b,3)-2*b*L2*R-b*R_2+b*a_2+sqrt(-a_2*(-
            R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*L2-L1_2)*(-R_2+a_2-2*L2*R-
            L2_2+b_2-2*L1*R-2*L1*L2-L1_2)))/(b_2+a_2)/L1/2.0;
        ///y2,x2: argumentos del atan2() para hallar q2
        y2 = (a_2+b_2-2*L2*R-L2_2-R_2-L1_2)/L1/(L2+R);
        x2 = -sqrt(-a_2*(-R_2+a_2-2*L2*R-L2_2+b_2+2*L1*R+2*L1*L2-L1_2)*(-
            R_2+a_2-2*L2*R-L2_2+b_2-2*L1*R-2*L1*L2-L1_2))/a/L1/(L2+R);
        }
    /// Angulos que se tendrÃ¡n en el punto de destino (simples variables
         intermedias)
    float q1 = atan2f(y1,x1);
    float q2 = atan2f(y2,x2);
    ROS_INFO("Coord Articulares de aproximacion: \nq1: %f, q2: %f",q1,q2)
        ; // Solo son variables intermedias para el calculo de X,Y de
        aprox
    /// Posicion de orientacion, MEDIDO RESPECTO AL Sref DEL BRAZO_1
    geometry_msgs::Point destino;
    destino.y = a + R * cos(M_PI + q1 - q2); //M_PI : Macro definida en
        ros/ros.h (#define _USE_MATH_DEFINES)
    destino.z = b + R * sin(M_PI + q1 - q2);
    /// Transformacion del punto objetivo desde el Sref del brazo_1 al
        Sref de la Base
    destino.y = destino.y + 0.026;
    destino.z = destino.z - 0.055;
    return destino;
    }

// Funcion auxiliar: Calculo pendiente recta tangente
float calcula_pendiente(float R, geometry_msgs::Point pos_obj)
    {
    ///Posicion actual
    // Equivalente a resolver la cinematica directa en la posicion actual
        . Lo leo por sensor de Gazebo para simplificar el problema
    float ya = pos_sensor.y;
    float za = pos_sensor.z;
    ///Posicion objeto (centro circunferencia)
    float y0 = pos_obj.y;
    float z0 = pos_obj.z;
    ///Magnitudes al cuadrado
    float R_2 = pow(R,2);
    float y0_2 = pow(y0,2);
    float z0_2 = pow(z0,2);
    float ya_2 = pow(ya,2);
    float za_2 = pow(za,2);
    float m = (z0*ya-y0*z0+y0*za-ya*za+R*sqrt(-R_2-2*z0*za-2*y0*ya+z0_2+
        y0_2+za_2+ya_2))/(-y0_2+2*y0*ya+R_2-ya_2);
    return m;
    }

// Funcion auxiliar: Funcion espera hasta alcanzar posicion deseada que
```

```
    se le envie
void esperar_alcanzar_pos(geometry_msgs::Point __pos_des)
    {
    int cont=0;
    while( cont < 1000 ) // Evitamos que sea por sobreoscilacion.
        Equivale a que este al menos 1 segundo, dado que se aumenta el
        contador cada milisegundo
        {
        //ROS_INFO("Posicion leida por el sensor: \n x: %f y: %f z: %f",
            pos_sensor.x, pos_sensor.y, pos_sensor.z);
        ros::Duration(0,100000).sleep(); // Se duerme durante un
            milisegundo para no sobrecargar la CPU y tampoco afectar al
            ciclo
        ros::spinOnce(); // Lectura del topic de posicion actual (
            pos_sensor), ya que al estar dentro de esta funcion no se
            atiende a las demas, pero tamb se deja de atender o leer el
            topic de la pos_sensor
        if((pos_sensor.y <= __pos_des.y + 0.0001)&&(pos_sensor.y >=
            __pos_des.y - 0.0001) && (pos_sensor.z <= __pos_des.z +
            0.0001)&&(pos_sensor.z >= __pos_des.z - 0.0001))
           {
            cont=cont+1; // Si estamos en el entorno a 0.1 milimetro se
                incrementa el contador
           }
        }
    ROS_INFO("Ultima posicion leida por el sensor: \n x: %f y: %f z: %f"
        , pos_sensor.x, pos_sensor.y, pos_sensor.z);
    }

// Funcion Topic lectura posicion actual
void pos_actual_efector(const nav_msgs::Odometry odom)
    {
     /// POSICION actual del efector (Respecto al sistema de referencia
         del Conjunto_base)
     pos_sensor = odom.pose.pose.position;
    }

// Funcion Servicio Objetivo
bool funcion_objetivo(manipulador::objetivo::Request& req,
manipulador::objetivo::Response& resp)
    {
    // EJEMPLO LLAMADA: $ rosservice call /servicio_objetivo '{objetivo:
        {tipo_objetivo: ir, posicion: {x: 0.0, y: 0.204, z: -0.264},
        orientacion: -0.26}}'
    // POSICION INICIAL: posicion: {x: 0.0, y: 0.204, z: -0.264}

    ROS_INFO("Localizacion actual: \nPosicion: x: %f y: %f z: %f",
        pos_sensor.x, pos_sensor.y, pos_sensor.z);
    tipo_objetivo = req.objetivo.tipo_objetivo; // Tipo de accion u
        objetivo que se esta recibiendo
    pos_obj = req.objetivo.posicion; // Posicion posicion deseada donde
        se quiere ir o se encuentra un objeto a coger o se desea soltar
        el objeto
    orientacion = req.objetivo.orientacion; // Orientacion que debe
        adoptar el efector o con la que se encuentra el objeto a coger o
```

```
    con la que se desea soltar el objeto
ROS_INFO("\nTipo de objetivo: %s \nPosicion: x: %f y: %f z: %f \
    nOrientacion: %f ", tipo_objetivo.c_str(), pos_obj.x, pos_obj.y,
    pos_obj.z, req.objetivo.orientacion);

if(!comprueba_espacio_de_trabajo(pos_obj))
  {
   ROS_INFO("Posicion deseada fuera del espacio de trabajo");
   return false;
   resp.confirmacion = false;
  }
if(!comprueba_orientacion(orientacion))
  {
   ROS_INFO("Orientacion deseada fuera del rango de la Articulacion_3
       ");
   return false;
   resp.confirmacion = false;
  }

// TIPO DE OBJETIVO "IR":
if(tipo_objetivo == "ir")
  {
    /// Envio Posicion deseada efector
    srv1.request.posicion = pos_obj;
    if (client1.call(srv1)) //Llamada al servicio
        {
         ROS_INFO("Llamada correcta al servicio --> Manipulador yendo
             al objetivo: \n[%f, %f, %f]",srv1.request.posicion.x ,
             srv1.request.posicion.y , srv1.request.posicion.z);
        }
    else
        {
         ROS_INFO("Error en la llamada al servicio posicion efector");
         return false;
         resp.confirmacion = false;
        }
    /// Envio Orientacion deseada efector
    srv3.request.consigna = orientacion;
    if (client3.call(srv3)) //Llamada al servicio
        {
         ROS_INFO("Llamada correcta al servicio --> Orientando efector
             ");
        }
    else
        {
         ROS_INFO("Error en la llamada al servicio orientacion efector
             ");
         return false;
         resp.confirmacion = false;
        }
  }

// TIPO DE OBJETIVO "COGER":
if(tipo_objetivo == "coger")
  {
```

```
///Se envia la barra a la posicion objetivo y con su orientacion
    (+ pi/2 ya que la barra se lanza vertical)
barra.request.initial_pose.position = pos_obj;
barra.request.initial_pose.orientation = tf::
    createQuaternionMsgFromRollPitchYaw(0, orientacion +
    1.570796327 ,0); ///geometry_msgs::Quaternion
    createQuaternionMsgFromRollPitchYaw(double roll,double pitch,
    double yaw) De "tf/transform_datatypes.h"
if(client_spawn.call(barra))
  {
     ROS_INFO("Llamada correcta a Spawn Barra en Gazebo");
  }
else
  {
     ROS_INFO("Fallo en la llamada a Spawn Barra en Gazebo");
  }
/// Calculo posicion de aproximacion orientada (con efector
    orientado en cuanto a roll). Punto en la frontera del volumen
    de seguridad donde el efector tiene la orientacion exacta para
     coger el objeto sin colisionar con el.
geometry_msgs::Point pos_aprox = calcula_punto_orientado(0.03,
    pos_obj); // R = 30 cm = radio del volumen de seguridad
ROS_INFO("Posicion de aproximacion (Sref base): \nx: %f y: %f z:
    %f", pos_aprox.x, pos_aprox.y, pos_aprox.z);

if(!comprueba_espacio_de_trabajo(pos_aprox))
  {
   ROS_INFO("Posicion de aprox fuera del espacio de trabajo");
   return false;
   resp.confirmacion = false;
  }

/// CASO SITUACION 2 <--> Estamos a la derecha de pos_aprox.y
if(pos_sensor.y>pos_aprox.y)
  {
   /// Calculo esa posicion de aprox tangente
    float m = calcula_pendiente(0.03,pos_obj); // Pendiente recta
        tangente al vol seg desde la posicion actual
    float y1 = pos_aprox.y;
    float z1 = pos_sensor.z + m*( pos_aprox.y - pos_sensor.y ); //
         Z de la interseccion de la recta tangente con la recta
        vertical en la posicion de aproximacion orientada
    /// Relleno un objeto del tipo Point con ella
    geometry_msgs::Point pos_aprox_tang;
    pos_aprox_tang.x = 0;
    pos_aprox_tang.y = y1;
    pos_aprox_tang.z = z1;
    /// Compruebo que esta dentro del espacio de trabajo
    if(!comprueba_espacio_de_trabajo(pos_aprox_tang))
     {
      ROS_INFO("Posicion de aprox tangente fuera del espacio de
          trabajo");
      return false;
      resp.confirmacion = false;
     }
```

```
    /// Envio a efector la posicion de aprox por tangente
    srv1.request.posicion = pos_aprox_tang;
    if (client1.call(srv1)) //Llamada al servicio
      {
       ROS_INFO("Llamada correcta al servicio --> Manipulador
           yendo a la posicion de aproximacion por tangente: [%f,%
           f,%f]", srv1.request.posicion.x , srv1.request.posicion
           .y , srv1.request.posicion.z );
      }
    else
      {
        ROS_INFO("Error en la llamada al servicio posicion efector
            ");
        return false;
        resp.confirmacion = false;
      }
    ///Espera a alcanzar posicion de aproximacion (valido en el
        entorno a 0.1 mm)
    esperar_alcanzar_pos(pos_aprox_tang);
  }
/// Envio a efector a posicion de aprox orientada
srv1.request.posicion = pos_aprox;
if (client1.call(srv1)) //Llamada al servicio
  {
    ROS_INFO("Llamada correcta al servicio --> Manipulador yendo a
         la posicion de aproximacion orientado: [%f,%f,%f]", srv1
        .request.posicion.x , srv1.request.posicion.y , srv1.
        request.posicion.z );
  }
else
  {
    ROS_INFO("Error en la llamada al servicio posicion efector");
    return false;
    resp.confirmacion = false;
  }
/// Orientacion en pitch (tercera articulacion)
srv3.request.consigna = orientacion;
if (client3.call(srv3)) //Llamada al servicio
  {
    ROS_INFO("Llamada correcta al servicio --> Orientando efector"
        );
  }
else
  {
    ROS_INFO("Error en la llamada al servicio orientacion efector"
        );
    return false;
    resp.confirmacion = false;
  }
///Espera a alcanzar posicion de aproximacion (valido en el
    entorno a 0.1 mm) antes de abrir la pinza
esperar_alcanzar_pos(pos_aprox);
///Apertura del efector
srv2.request.consigna = -1;
if (client2.call(srv2)) //Llamada al servicio
```

```
        {
          ROS_INFO("Llamada correcta al servicio --> Abriendo efector");
        }
      else
        {
          ROS_INFO("Error en la llamada al servicio de apertura efector"
              );
          return false;
          resp.confirmacion = false;
        }
      /// Esperamos un segundo, que es tiempo de sobra para que el
          efector quede abierto y conseguir estailidad
      ros::Duration(2,0).sleep();
      /// Aproximacion final al objeto (Caso parametrico, dividiendo el
           trayecto en 5 pasos)
      ROS_INFO("Manipulador yendo al punto final deseado : [%f,%f,%f]",
           pos_obj.x , pos_obj.y , pos_obj.z );
      float y0 = pos_aprox.y;
      float z0 = pos_aprox.z;
      float Vy = pos_obj.y - pos_aprox.y;
      float Vz = pos_obj.z - pos_aprox.z;
      float t=0.2;
      while(t<=1)
         {
          srv1.request.posicion.y = y0 + t*Vy;
          srv1.request.posicion.z = z0 + t*Vz;
          t = t + 0.2;
          client1.call(srv1);
          ros::Duration(0,1e8).sleep(); //10 elevado a 8 nanosegundos
         }
      ///Espera a alcanzar posicion objetivo (valido en el entorno a
          0.1 mm)
      esperar_alcanzar_pos(pos_obj);
      ///Cierre del efector --> Objeto cogido
      srv2.request.consigna = 1;
      if (client2.call(srv2)) //Llamada al servicio
        {
          ROS_INFO("Llamada correcta al servicio --> Cerrando efector");
        }
      else
        {
          ROS_INFO("Error en la llamada al servicio de cierre efector");
          return false;
          resp.confirmacion = false;
        }
    }

  // TIPO DE OBJETIVO "SOLTAR":
  if(tipo_objetivo == "soltar")
      {
       /// Calculo posicion de aproximacion orientada (con efector
           orientado en cuanto a roll).
       geometry_msgs::Point pos_aprox = calcula_punto_orientado(0.03,
           pos_obj); // R = 30 cm = radio del volumen de seguridad
       ROS_INFO("Posicion de aproximacion (Sref base): \nx: %f y: %f z:
```

```
      %f", pos_aprox.x, pos_aprox.y, pos_aprox.z);
 if(!comprueba_espacio_de_trabajo(pos_aprox))
    {
      ROS_INFO("Posicion de aprox fuera del espacio de trabajo");
      return false;
      resp.confirmacion = false;
    }
/// Envio efector a posicion de aprox orientada
srv1.request.posicion = pos_aprox;
if (client1.call(srv1)) //Llamada al servicio
  {
    ROS_INFO("Llamada correcta al servicio --> Manipulador yendo a
         la posicion de aproximacion orientado: [%f,%f,%f]", srv1
        .request.posicion.x , srv1.request.posicion.y , srv1.
        request.posicion.z );
  }
else
  {
    ROS_INFO("Error en la llamada al servicio posicion efector");
    return false;
    resp.confirmacion = false;
  }
/// Orientacion en pitch (tercera articulacion)
srv3.request.consigna = orientacion;
if (client3.call(srv3)) //Llamada al servicio
  {
    ROS_INFO("Llamada correcta al servicio --> Orientando efector"
        );
  }
else
  {
    ROS_INFO("Error en la llamada al servicio orientacion efector"
        );
    return false;
    resp.confirmacion = false;
  }
///Espera a alcanzar posicion de aproximacion (valido en el
    entorno a 0.1 mm) antes de abrir la pinza
esperar_alcanzar_pos(pos_aprox);
///Aproximacion final al objetivo
srv1.request.posicion = pos_obj;
if (client1.call(srv1)) //Llamada al servicio
  {
    ROS_INFO("Llamada correcta al servicio --> Manipulador yendo
        al punto final deseado : [%f,%f,%f]", srv1.request.
        posicion.x , srv1.request.posicion.y , srv1.request.
        posicion.z );
  }
else
  {
    ROS_INFO("Error en la llamada al servicio posicion efector");
    return false;
    resp.confirmacion = false;
  }
///Espera a alcanzar posicion de objetivo (valido en el entorno a
```

```
            0.1 mm)
        esperar_alcanzar_pos(pos_obj);
        ///Esperamos a que se realize el proceso de soldado o el que se
            estime oportuno (suponemos que llevara 5 segundos)
        ros::Duration(5,0).sleep();
        ///Apertura del efector
        srv2.request.consigna = -1;
        if (client2.call(srv2)) //Llamada al servicio
          {
            ROS_INFO("Llamada correcta al servicio --> Abriendo efector");
          }
        else
          {
            ROS_INFO("Error en la llamada al servicio de apertura efector"
                );
            return false;
            resp.confirmacion = false;
          }
        /// Vuelta del efector a posicion de aprox orientada. En este
            caso si tiene que ser obligatoriamente esta posicion para
            dejar la barra sin colisionar con ella
        srv1.request.posicion = pos_aprox;
        if (client1.call(srv1)) //Llamada al servicio
          {
            ROS_INFO("Llamada correcta al servicio --> Manipulador yendo a
                 la posicion de aproximacion orientado: [%f,%f,%f]", srv1
                .request.posicion.x , srv1.request.posicion.y , srv1.
                request.posicion.z );
          }
        else
          {
            ROS_INFO("Error en la llamada al servicio posicion efector");
            return false;
            resp.confirmacion = false;
          }
        }
    resp.confirmacion = true;
    return true;
    }

// Funcion Servicio de Reset: Elimina la barra y devuelve el efector a la
     posicion inicial
bool reset(std_srvs::Empty::Request& req, std_srvs::Empty::Response&
resp)
  {
    ROS_INFO("\n\n-----------RESET PLANIFICACION----------- \n\n");
    ///Elimina barra
    elimina_barra.request.model_name = "barra_sin_gravedad";
    client_elimina_barra.call(elimina_barra);
    ///Pinza a posicion de equilibrio
    srv2.request.consigna = 0;
    client2.call(srv2);
    ///Efector a Posicion inicial
    srv1.request.posicion.x = 0;
    srv1.request.posicion.y = 0.204;
```

```
    srv1.request.posicion.z = -0.264;
    client1.call(srv1);
    /// Orientacion en pitch (tercera articulacion)
    srv3.request.consigna = 0;
    client3.call(srv3);
  }

int main(int argc, char **argv) {
  // Creacion del nodo con su nombre y etiqueta nodehandle
  ros::init(argc, argv, "planificacion_manipulador");
  ros::NodeHandle na;
  // Espera a que empieze a emitir /clock
  ros::Time::waitForValid();
  ROS_INFO("Entra el nodo planificacion_manipulador");
  // Cliente al servicio Pos des efector:
  client1 = na.serviceClient<control_cartesiano_manipulador::
      Pos_des_efector>("/controlador_cartesiano/cambio_pos_efector");
  // Cliente al servicio apertura/cierre efector:
  client2 = na.serviceClient<control_cartesiano_manipulador::
      Cambio_Consigna>("/controlador_cartesiano/cambio_gancho_efector");
  // Cliente al servicio cambio pitch del efector:
  client3 = na.serviceClient<control_cartesiano_manipulador::
      Cambio_Consigna>("/controlador_cartesiano/cambio_pitch_efector");
  // Cliente al servicio spawn_urdf_model (enviar URDFs a Gazebo):
  client_spawn = na.serviceClient<gazebo_msgs::SpawnModel>("/gazebo/
      spawn_urdf_model");
  // Se rellenan los campos que no van a variar, es decir, todos a
      excepcion de las posiciones iniciales y la orientacion que vendran
      impuestas por las que lleguen en pos_obj
      // Se pasa el contenido del URDF a un STRING y se copia al campo
          model_xml del servicio de envio de robots a Gazebo
      std::ifstream file("/home/riqui/ros_workspace/manipulador/robots/
          barra_sin_gravedad.urdf");
      std::string line;
      while(!file.eof())
        {
          std::getline(file,line);
          barra.request.model_xml+=line;
        }
      file.close();
      // Se rellenan los demas campos
      barra.request.model_name = "barra_sin_gravedad";
      barra.request.robot_namespace = "barra_sin_gravedad";
      barra.request.reference_frame = "manipulador::table_top_link"; //Es
           equivalente al sistema de referencia del conjunto base
  // Cliente al servicio para eliminar modelos de Gazebo
  client_elimina_barra = na.serviceClient<gazebo_msgs::DeleteModel>("/
      gazebo/delete_model");
  // Subscripcion topic Posicion actual del efector en Gazebo
  ros::Subscriber Sub_pos_actual = na.subscribe("/sensor_pos_efector", 1,
       pos_actual_efector);
  // Servicio para recepcion de objetivo
  ros::ServiceServer server_1 = na.advertiseService ("/servicio_objetivo"
      , funcion_objetivo); // Objeto servicio
  // Servicio reseteo planificacion
```

```
  ros::ServiceServer server_reset = na.advertiseService("/reset", reset);
  ros::Duration(5,0).sleep();

  ros::spin(); // Espera hasta !ros::ok() --> El programa permanecera
      abierto leyendo topics y ofreciendo estos servicios hasta que se
      cierre mediante [Ctrl] + [C]
  return 0;
}
```

F.5 control_cartesiano_manipulador.launch

```
<!--Carga los parametros del archivo yaml y ejecuta pr2_manager con
nuestro controlador-->
<launch>
   <rosparam file="$(find control_cartesiano_manipulador)/yaml/
       control_cartesiano_manipulador.yaml" command="load" />
   <node pkg="pr2_controller_manager" type="spawner" args="
       controlador_cartesiano" name="control_cartesiano_spawner" />
</launch>
```

F.6 man_mesa_gazebo_control_cartesiano.launch

```
<launch>
    <!--********** GAZEBO, URDF **********-->
    <!-- Usar tiempo de simulacion en vez del real-->
    <param name="/use_sim_time" value="true" />
    <!--Lanza Gazebo en mundo vacio-->
    <node name="gazebo" pkg="gazebo" type="gazebo" args="-u $(find
        gazebo_worlds)/worlds/empty.world" respawn="false" output="log"/>
    <!-- Enviamos el robot (manipulador) al servidor de parametros para
        que lo lea Gazebo -->
    <param name="robot_description" command="$(find xacro)/xacro.py $(
        find manipulador)/robots/man_mesa.urdf" />
    <!-- Lanzamos el robot (manipulador) con el nodo spawn_model de
        Gazebo. Los parametros son el nombre en el servidor de parametros
        , la localizacion inicial en Gazebo, ... -->
    <node name="spawn_man_mesa" pkg="gazebo" type="spawn_model" args="-
        urdf -param robot_description -x 0 -y 0 -z 0.51 -model
        manipulador" respawn="false" output="screen" />
    <!--********** CONTROLADOR **********-->
    <!-- Incluimos el lanzamiento de los controladores creados y
        definidos en el paquete control_cartesiano_manipulador-->
    <include file="$(find control_cartesiano_manipulador)/launch/
        control_cartesiano_manipulador.launch"/>
    <!--********** RXPLOT **********-->
    <!--Las 4 pos articulares juntas-->
    <node pkg="rxtools" type="rxplot" name="rxplot_articulaciones" args="
        -P -r 10 -l 'Articulacion_1','Articulacion_2','Articulacion_3','
        Articulacion_efector' -t POSICIONES_ARTICULACIONES /
        controlador_cartesiano/Lectura_datos_articulacion_1/position /
        controlador_cartesiano/Lectura_datos_articulacion_2/position /
        controlador_cartesiano/Lectura_datos_articulacion_3/position /
```

```
        controlador_cartesiano/Lectura_datos_articulacion_efector/
        position" />
    <!--Sensor posicion efector-->
    <node pkg="rxtools" type="rxplot" name="rxplot_pos_efector" args="-P
        -r 10 -l 'Y','Z' -t POSICION_EFECTOR /sensor_pos_efector/pose/
        pose/position/y /sensor_pos_efector/pose/pose/position/z" />
    <!--********** Monitorizacion: Consola **********-->
    <node pkg="rxtools" type="rxconsole" name="rxconsole" />
</launch>
```

Apéndice G

Códigos QuadRotor equipado con Manipulador

G.1 Propiedades.h

```
// Propiedades y constantes dinamicas
#define g 9.800
//#define m_quad 1.73
//#define m_man 0.371005
//#define m_bateria 0.231 Masa Total:
#define m 2.332005 //kg

// Las inercias son las del quadrotor solamente...
#define Ixx 0.01632655 // kg m2 equivalente a 1 N m s2 en que lo
expresan en el Chapter 3 #define Iyy 0.01632655 // kg m2 #define Izz
0.02642555 // kg m2

//Constantes aerodinamicas
#define b 54.2e-6 // N s2 #define d 1.1e-6 // N m s2 #define l 0.286
// m

// Coeficientes de ponderacion Contra-reacciones
#define C1x 1 #define C1y 1 #define C2x 0.95 #define C2y 0.45
```

G.2 control.cpp

Se añaden a continuación solo los trozos de código nuevos respecto al archivo *control.cpp* del paquete *modelquad*, teniendo en cuanta que, además de estos, las únicas modificaciones realizadas son los cambios de todas las referencias al paquete *modelquad* por el paquete *Quadrotor_manipulador*:

```
                    (...)

/// Tipo de mensaje informacion del controlador del manipulador
#include "Quadrotor_manipulador/Lectura_datos.h"
                    (...)

/// Momentos aplicados en la articulacion 1, descompuestos y anteriores
```

```
float M1x=0; float M1y=0; float M1x_ant=0; float M1y_ant=0;

/// Funcion de lectura del topic Lectura_datos_articulacion_1
void lectura_esfuerzos_brazo_1(const
Quadrotor_manipulador::Lectura_datos datos1)
    {
        /// Momento medido en la articulacion 1, que esta a 45 grados en
            el primer cuadrante
        float M1 = datos1.measured_effort;
        /// Momento en sentido positivo del eje X, es decir, roll>0
        M1x= M1/sqrt(2);
        /// Momento en sentido positivo del eje Y, es decir, pitch>0
        M1y= - M1/sqrt(2);

        //ROS_INFO("ENTRA LECTURA DATOS ARTICULACION 1");
        //ROS_INFO("M1: %f, M1x: %f, M1y: %f", M1, M1x, M1y);
    }

/// Momentos aplicados en la articulacion 2, descompuestos y anteriores
float M2x=0; float M2y=0; float M2x_ant=0; float M2y_ant=0;

/// Funcion de lectura del topic Lectura_datos_articulacion_2
void lectura_esfuerzos_brazo_2(const
Quadrotor_manipulador::Lectura_datos datos2)
    {
        float M2 = datos2.measured_effort;

        M2x= M2/sqrt(2);
        M2y= - M2/sqrt(2);

        //ROS_INFO("ENTRA LECTURA DATOS ARTICULACION 2");
        //ROS_INFO("M2: %f, M2x: %f, M2y: %f", M2, M2x, M2y);
    }

int main(int argc, char **argv)
    {
                        (...)
        /// Subscripcion a Topic para lectura del momento en la
            articulacion 1 del manipulador
        ros::Subscriber Sub_M1 = n.subscribe("/controlador_cartesiano/
            Lectura_datos_articulacion_1", 1, lectura_esfuerzos_brazo_1);
        /// Subscripcion a Topic para lectura del momento en la
            articulacion 2 del manipulador
        ros::Subscriber Sub_M2 = n.subscribe("/controlador_cartesiano/
            Lectura_datos_articulacion_2", 1, lectura_esfuerzos_brazo_2);
                        (...)
        /// Modelo Inverso (aceleraciones-->Fuerzas y momentos)
        /// Fuerza[N]
        U1act = m*(uzk + g)/(cos(rk)*cos(pk));
        /// Momentos[N m]
        U2act = urk*Ixx + (C2x*M2x + C1x*M1x); //roll-> -y
        U3act = upk*Iyy + (C2y*M2y + C1y*M1y); //pitch-> x
        U4act = uyk*Izz;
                        (...)
    }
```

G.3 Quad_man_control.launch

```
<launch>
<!--*************** SERVIDOR DE PARAMETROS Y GAZEBO *************-->
    <!-- Usar tiempo de simulacion en vez del real-->
    <param name="/use_sim_time" value="true" />
    <!--Lanza Gazebo en mundo vacio-->
    <node name="gazebo" pkg="gazebo" type="gazebo" args="-u $(find
        gazebo_worlds)/worlds/empty.world" respawn="false" output="log"/>
    <!-- Enviamos el robot al servidor de parametros para que lo lea
        Gazebo -->
    <param name="robot_description" command="$(find xacro)/xacro.py $(
        find Quadrotor_manipulador)/robots/man_quad.urdf" />
    <!-- Lanzamos el robot en Gazebo -->
    <node name="spawn_man_quad" pkg="gazebo" type="spawn_model" args="-
        urdf -param robot_description -x 0 -y 0 -z 0.5 -model man_quad"
        respawn="false" output="screen" />

 <!--***************** CONTROL MANIPULADOR ************************-->
    <!-- Incluimos el lanzamiento de los controladores del paquete
        control_cartesiano_manipulador (CONTROL CARTESIANO)-->
    <include file="$(find control_cartesiano_manipulador)/launch/
        control_cartesiano_manipulador.launch"/>

 <!--***************** CONTROL QUADROTOR ************************-->
    <!-- Ejecutamos el nodo de control de posicion y el de arranque (1 m)
         del quadrotor-->
    <node name="control" pkg="Quadrotor_manipulador" type="control"
        output="screen"/>
    <node name="arranque" pkg="Quadrotor_manipulador" type="arranque"
        output="screen"/>

 <!--******************** PLOTS ************************-->
    <!-- Posicion Quad-->
    <node pkg="rxtools" type="rxplot" name="rxplot_positionXYZ" args="-P
        -r 1 -l 'X[m]','Y[m]','Z[m]' -t PoscionQuad /Odometria/pose/pose/
        position/x /Odometria/pose/pose/position/y /Odometria/pose/pose/
        position/z" />
    <!-- Orientacion Quad-->
    <node pkg="rxtools" type="rxplot" name="rxplot_OrientacionRPY" args="
        -P -r 1 -l 'R[rad]','P[rad]','Y[rad]' -t OrientacionQuad '/
        Pos_RPY_actual/orientacion/x','/Pos_RPY_actual/orientacion/y','/
        Pos_RPY_actual/orientacion/z'" />
    <!-- Giros motores-->
    <node pkg="rxtools" type="rxplot" name="rxplot_giros_motores" args="-
        P -r 1 -l 'G1[rad/s]','G2[rad/s]','G3 [rad/s]','G4[rad/s]' -t
        Giros_Motores /Giros_Motores/G1 /Giros_Motores/G2 /Giros_Motores/
        G3 /Giros_Motores/G4" />
    <!--Sensor posicion efector-->
    <node pkg="rxtools" type="rxplot" name="rxplot_pos_efector" args="-P
        -r 10 -l 'Y','Z' -t PosicionEfector /sensor_pos_efector/pose/pose
        /position/y /sensor_pos_efector/pose/pose/position/z" />
    <!-- Lectura articulacion 1-->
    <node pkg="rxtools" type="rxplot" name="rxplot_articulacion_1" args="
        -P -r 1 -l 'Posicion','Velocidad','Esfuerzo comandado','Esfuerzo
```

```
        medido' -t Articulacion_1 /controlador_cartesiano/
        Lectura_datos_articulacion_1/position /controlador_cartesiano/
        Lectura_datos_articulacion_1/velocity /controlador_cartesiano/
        Lectura_datos_articulacion_1/commanded_effort /
        controlador_cartesiano/Lectura_datos_articulacion_1/
        measured_effort" />
    <!-- Lectura articulacion 2-->
    <node pkg="rxtools" type="rxplot" name="rxplot_articulacion_2" args="
        -P -r 1 -l 'Posicion','Velocidad','Esfuerzo comandado','Esfuerzo
        medido' -t Articulacion_2 /controlador_cartesiano/
        Lectura_datos_articulacion_2/position /controlador_cartesiano/
        Lectura_datos_articulacion_2/velocity /controlador_cartesiano/
        Lectura_datos_articulacion_2/commanded_effort /
        controlador_cartesiano/Lectura_datos_articulacion_2/
        measured_effort" />
    <!-- Lectura esfuerzos cobre el quad-->
    <node pkg="rxtools" type="rxplot" name="rxplot_esfuerzos_quad" args="
        -P -r 1 -l 'Mquadx[Nm]','Mquady[Nm]' -t Esfuerzos_quad '/
        Medida_Mquad/wrench/torque/x','/Medida_Mquad/wrench/torque/y'" />

  <!-- ********** CONSOLA **********-->
  <node pkg="rxtools" type="rxconsole" name="rxconsole" />

</launch>
```