2019

ALVARADO GALICIA FELIPE FONSECA CAMARENA JONATHAN

PROGRAMACIÓN DE ROBOTS INDUSTRIALES

30-5-2019

EV\_2\_2\_manipulador industrial de robots en modo manual

UPZMG



**OBJETIVO:**

Nuestro objetivo general en esta práctica es conocer el entorno de la programación en los robots y saber ¿cuáles programas vamos a utilizar? Para así practicar y ser autoritarios con base a lo aprendido.

MATERIALES:

Simulador de software de robots

**MARCO TEORICO:**

Entornos de Simulación de Robots

A la hora de afrontar el diseño y construcción de un robot, se hace necesario disponer de mecanismos capaces de realizar una simulación lo más fiel posible de entornos reales. El elevado coste de la construcción puede ser tan alto que podría hacer inabordable la construcción de nuestro proyecto.

***Requisitos del software analizado***

Morfología del robot

Un brazo robótico está formado por varios elementos como son la estructura mecánica, transmisiones, sistemas de accionamiento, sistemas sensoriales, sistemas de control y potencia, y elementos terminales. Además, mecánicamente el robot está compuesto de eslabones unidos mediante articulaciones. Todos estos elementos tienen una dimensiones y comportamientos físico que conforman el funcionamiento del robot.

Cinemática del robot

Una vez definido como será el robot físicamente, el simulador deberá permitir y controlar la cinemática propia del robot para evitar movimientos imposibles. El simulador deberá permitir mover tanto los eslabones como las articulaciones respetando la cadena cinemática. Además, se deberá respetar en todo momento las posibles restricciones de movimiento definidos en cada eje.

Dinámica del robot

La Dinámica del robot relaciona el movimiento del robot y las fuerzas implicadas en el mimo. En este sentido, el simulador deberá permitir definir los valores propios a cada elemento del robot. Deberá tener en cuenta todas las fuerzas implicadas en el sistema y deberá actuar en consecuencia ante movimientos o cargas elevadas.

Entorno

Como se ha indicado anteriormente, el sistema debería permitir definir el entorno en el que opera el robot y además debería disponer de la posibilidad de añadir sensores que respondan ante las condiciones indicadas en la simulación.

Para poder definir bien el entorno se debería permitir modificar:

• Sistemas de física: Por norma general la física aplicada en el brazo robótico estará definido entre otros por la fuerza de gravedad, pero sería interesante poder modificar estos valores para cambiar el entorno de trabajo a situaciones más extremas. Por ejemplo, se podría eliminar la gravedad para comprobar el funcionamiento en entornos ingrávidos como por ejemplo el espacio, o aumentar el nivel de atmósfera para simular situaciones submarinas. Otros valores que se podrían modificar serían los valores de humedad y temperatura (por ejemplo) que deberían afectar al movimiento de los motores y actuadores.

• Sistemas de renderizado 3d: El sistema de renderizado 3D es otro punto a tener en cuenta y sería interesante que el renderizado sea lo más realista posible sin que por ello el sistema se vea ralentizado.

• Sistemas de modelado: Este es un detalle menor, pero la existencia de un sistema de modelado dentro del propio simulado facilitaría la construcción y modificación del robot. Si pensamos en que estamos creando un robot, es posible que para ciertos movimientos las propias piezas choquen entre sí, y si el simulador permitiese realizar el ajuste en la propia interface ganaríamos agilidad. Al menos el simulador debería admitir modelos 3D en los formatos estándar.

• Entorno de diseñado y programación: Otro punto a tener en cuenta a la hora de analizar el simulador es si la ejecución del mismo se realiza desde un GUI (Graphical User Interface – Interfaz gráfica de usurio) o por el contrario la simulación se programa completamente con scripts y luego se lanza la simulación por línea de comandos.

Programación

En este punto diferenciaré dos tipos de programaciones:

• Sistema: Un buen simulador debería permitir la creación de nuevos módulos para implementar aquellas situaciones o comportamientos no contemplados en el propio simulador. El sistema debería disponer de un API lo suficientemente amplio para permitir la programación de estos añadidos.

• Robot: No podemos olvidar que uno de los objetivos del simulador es el de llevar a la realidad las pruebas realizadas. La mayoría de los robots modernos son programables y por tanto disponen de la capacidad de cargar ficheros con el código a ejecutar. El simulador debería permitir realizar y exportar el movimiento empleado en la simulación en un lenguaje comprensible por el robot real, o al menos a un lenguaje que permita su conversión sobre un software especializado.

Conexión

Aunque se trate de un simulador, es importante que el sistema simulado funcione como un sistema real. En este sentido, el robot debería trabajar de forma autónoma, comunicándose con el sistema de control mediante alguno de los protocolos más habituales. En este sentido se verán que tipos de protocolos se utiliza en cada simulador.

Simulador elegido



Gazebo

Gazebo es un software de simulación y su principal ventaja, además de disponer de una licencia Apache 2 (y por tanto libre), es que está especialmente diseñado para probar de forma rápida algoritmos y diseño de robots.

http://gazebosim.org

Morfología del robot

Para la parte de renderizado, Gazebo cuenta con la ayuda de OGRE. Este software ofrece un entorno sencillo para el diseño orientado a objetos e independiente de la implementación 3D, por lo que utiliza tanto DirectX como OpenGL.

Gazebo dispone de un editor que permite añadir figuras básicas como cilindros, esferas o cubos, u otras más elaboradas basadas en gráficos SVG extruibles o mayas 3D en formato “. dae” o “. stl”. Para crear este tipo de mayas se puede utilizar software libre como por ejemplo Blender.

Para unir las figuras y conformar el robot, dispones de uniones en las que se puede definir el tipo de movimiento permitido (Rotación, Prismático, esférico, etc). Además, permite definir los límites de movimiento, límites de fuerza soportada y velocidad soportada, viscosidad, fricción, etc.

Otra posibilidad que ofrece Gazebo es la de importar robots mediante el formato SDF. Este formato basado en XML describe objetos y entornos para simulación de robots, visualización y control. Además de permitir desarrollar nuestros propios prototipos, el sistema dispone de modelos propios.

Dinámica y Cinemática

Como se ha visto en el apartado anterior, es posible la parametrización de los diferentes elementos que intervienen en el robot. Además, Gazebo dispone de un conjunto amplio de clases matemáticas que permite definir y operar por ejemplo con matrices, quaternios, planos, etc.

En cuanto a la dinámica, es compatible con varios motores físicos como:

• ODE (Open Dynamics Engine) que es una librería para la simulación de cuerpos rígidos y permite el control de colisiones y rozamientos.

• SIMBODY. Esta librería dota a nuestros objetos de capacidades dinámicas, resolviendo por ejemplo la segunda ley de Newton (F=m\*a) (Fuerza es igual a la mása por la aceleración).

• DART (Dynamyc Animation and Robotics Toolkit). Esta librería ofrece estructuras de datos y algoritmos para cinemática y dinámica.

• BULLET. Esta librería de Código Abierto es un motor de detección 3D y dinámica de cuerpos tanto rígidos como blandos.

Entorno

Gazebo dispone de una interface en el que se permite diseñar todo el entorno en el que operará el robot, dotando a todos los objetos las características de cuerpos rígidos con colisión. Además de colisiones propias de los objetos, se dispone de la posibilidad de añadir sensores de tipo laser, cámaras 2D o 3D, sensores de fuerza, de contacto, etc. Todos estos sensores pueden ser incluidos con ruido, con lo que se consigue una simulación más real.

Otro punto a favor, es la posibilidad de realizar simulaciones remotas y en la nube.

Programación

En cuanto a la programación del sistema, y al tratarse de un software de código abierto, se dispone de la posibilidad de desarrollar plugins específicos que alteren el comportamiento del robot, de los sensores o del entorno, posibilitando la configuración de cualquier situación.

Para ello dispone de un API muy completo de programación en C++.

En cuanto a la generación de programas para implantarlos directamente en robots, no parece que exista directamente esta posibilidad.

Instalación y pruebas de simulación

Aunque el sistema a partir de la versión 6.0 es compatible con Windows, no ha conseguido (aún) instalarlo correctamente en este Sistema Operativo. Sin embargo, la instalación en una máquina con Linux (concretamente en una distribución Ubuntu) es sencilla y automática.

Robótica: Cinemática y Dinámica del Robot

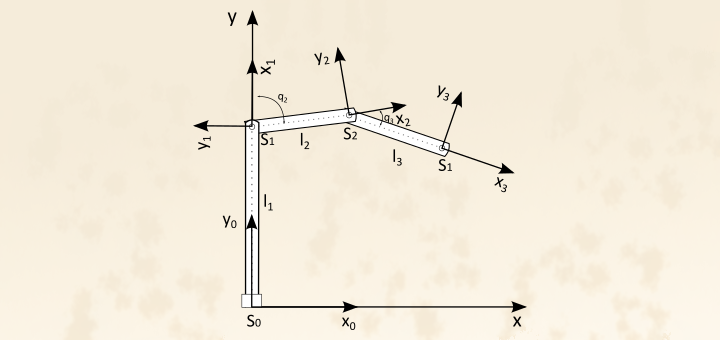
Se expondrá dos conceptos fundamentales que deberemos tener en cuenta en la construcción de nuestro brazo robótico. Estos conceptos son la cinemática y la dinámica.

En esta ocasión, el dato es divulgativo y por tanto no me centraré en aspectos formales y en fórmulas matemáticas, sin embargo, en 7mo, entraremos a detallar estos temas con un mayor nivel de complejidad.

Estructura de un robot

Antes de empezar a definir el movimiento de un brazo robótico, deberemos ver cómo está estructurado el robot.

Podemos definir un brazo robótico como una cadena de elementos denominados eslabones que se unen mediante articulaciones.



Estas articulaciones darán movimiento a nuestro brazo y se dividen en:

• Movimientos prismáticos: Son movimientos de desplazamiento y típicamente en una única dimensión (hacia delante o atrás, arriba o abajo, izquierda o derecha).

• Movimientos rotacionales: Son movimientos de giro en una de las dimensiones (eje x, y o z).

Además de estos movimientos básicos podemos encontrar movimientos más complejos como los movimientos en esfera o “rótula” los cuales permiten una rotación en las tres dimensiones, movimientos planares que permiten el movimiento en dos dimensiones a la vez o movimientos cilíndricos que permiten un movimiento rotacional y prismático a la vez.

Independientemente del tipo de articulación que utilicemos, lo habitual es nombrar a cada movimiento en una única dimensión mediante el término “Grado de Libertad” (GDL). Con esto podemos definir nuestro robot con base al número de grados de libertad y que para la figura 1 del ejemplo tendríamos un robot con 3 grados de libertad. En el caso de una articulación de rótula tendríamos también 3 grados de libertad (Una rotación en el eje X, otra en el eje Y, y una última en el eje Z).

Para aclarar un poco más el asunto, existen varias formas de definir un robot, siendo el número de grados de libertad uno de ellos, pero también podríamos por su configuración o por su generación. Por otro lado, tenemos que tener claro que cada articulación será movida por un actuador, que para resumir será un motor.

Dinámica del robot

La dinámica se centra en la fuerza necesaria para realizar el movimiento. Para ello se han de tener en cuenta detalles tan importantes como la fricción, la potencia de los motores, la gravedad, la masa del propio brazo robótico o el objeto que deseemos mover.

Para realizar el movimiento con la suficiente suavidad, se suelen utilizar controladores PID. Estos controladores disponen de 3 tipos de parámetros:

• Proporcional (P)

• Integral (I)

• Derivado (D)

**PROCEDIMIENTO:**

Simulación de Robots con Gazebo. Construcción del robot

Veremos cómo montar nuestro robot.

https://youtu.be/uI406H8YxCA

Antes de comenzar con el desarrollo del robot, deberemos configurar nuestro entorno de trabajo. Para ello deberemos crear una carpeta en la que alojaremos nuestro código y en la que estarán localizadas las librerías de compilación.

El primer paso será crear las carpetas mediante el comando “mkdir”. En nuestro caso escogeremos la carpeta de raíz de usuario “~/”. La tilde de la ñ se obtiene con la combinación de teclas alt+124 o pulsando alt gr + 4. Otra opción es poner /home/”usuario” (lo que va en comillas, es el nombre que cada quien le puso de usuario a su sistema Linux)

Crearemos una carpeta inicial (yo la llamare “Robot”) y dentro de esta crearemos la carpeta “src” (de source que significa fuente en inglés).

Ahora deberemos configurar el terminal para que conozca la ubicación de ROS, para ello ejecutaremos el comando “source /opt/ros/kinetic/setup.bash”.

Ahora inicializamos catkin y compilamos.

cd ~/Robot/src/

catkin\_init\_workspace

cd ..

catkin\_make

Ahora ya estamos en disposición de comenzar a desarrollar nuestro robot. Para ello empezaremos creando dentro de src nuestro proyecto. Esto lo hacemos utilizando el comando “catkin\_create\_pkg” en el que podemos indicar las librerías que vallamos a necesitar. En nuestro caso necesitaremos:

• gazebo\_plugins

• gazebo\_ros

• gazebo\_ros\_control

• ros\_controllers

• roscpp

• std\_msgs

El siguiente paso es construir nuestro mundo. Mi recomendación es que lances Gazebo y directamente guardes un mundo en blanco. Para ello crea una carpeta en el que guardar el mundo y dale un nombre reconocible. En mi caso lo he llamado “miRobot.world” y lo he guardado en la carpeta “worlds”.

Para lanzar nuestro proyecto, utilizaremos el comando “roslaunch”. Este comando necesita un fichero de configuración en el que indicaremos como queremos lanzar nuestra simulación. El fichero lo crearemos en una carpeta dentro la carpeta “worlds” y con el nombre “miRobot.launch”

<launch>

                <arg name="paused" default="false"/>

                <arg name="use\_sim\_time" default="true"/>

                <arg name="gui" default="true"/>

                <arg name="headless" default="false"/>

                <arg name="debug" default="false"/>

                <include file="$(find gazebo\_ros)/launch/empty\_world.launch">

                               <arg name="world\_name" value="$(find miRobot)/worlds/miRobot.world"/>

                               <arg name="verbose" value="true"/>

                               <arg name="debug" value="$(arg debug)" />

                               <arg name="gui" value="$(arg gui)" />

                               <arg name="paused" value="$(arg paused)"/>

                               <arg name="use\_sim\_time" value="$(arg use\_sim\_time)"/>

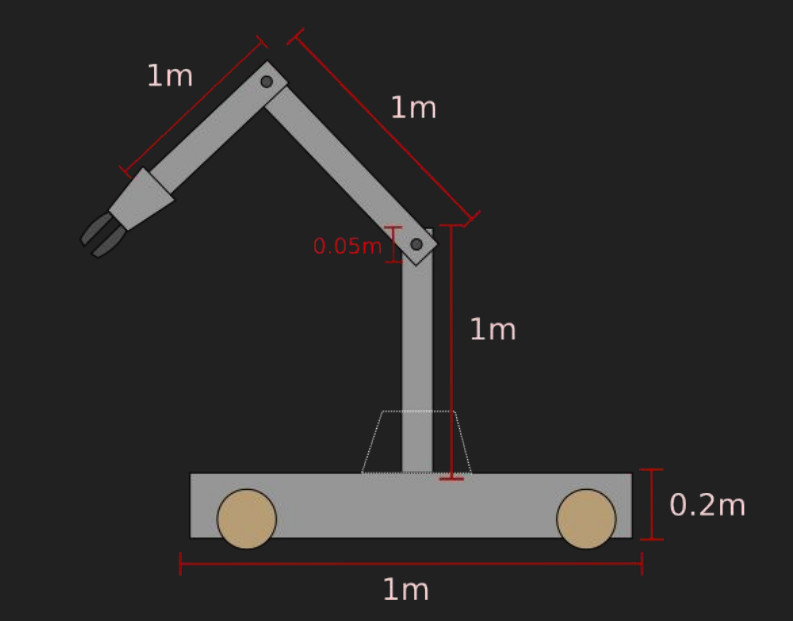
                               <arg name="headless" value="$(arg headless)"/>

                </include>

</launch>

Con las opciones anteriores arrancaremos nuestro robot con el mundo pausado de tal forma que no se aplicará movimiento y físicas a nuestro robot.

Definición del robot



Para definir el robot utilizaremos el lenguaje SDF (http://www.sdformat.org). En este lenguaje se definirá el robot mediante el elemento Model.

Dentro de este elemento se utilizarán, básicamente, otros 2 elementos, el enlace y la unión.

Link

El link permite definir el cómo se deberá mostrar el eslabón, cuáles son los criterios de colisión y cuál es la dinámica aplicada.

En el siguiente trozo del código de definición se puede ver:

• La posición que ocupa el brazo “Pose”.

• La inercia que tiene junto con su masa.

• El área de colisión. Este puede ser un objeto básico que encierre al objeto visual, o en nuestro caso es un modelo un poco más elaborado. Además de la geometría de colisión, se encuentran parámetros de superficie como la fricción, el contacto, etc.

• Otro dato importante define como se deberá visualizar el elemento. Esto se encuentra en el tag Visual.

Todos estos parámetros son procesados por Ogre (http://www.ogre3d.org/about/features) y por ODE (http://www.ode.org/) para simular un entorno real.

<link name='arm\_link\_0'>

  <pose frame=''>0.143 0 0.046 0 -0 0</pose>

  <inertial>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <inertia>

            <ixx>0.01</ixx>

            <ixy>0</ixy>

            <ixz>0</ixz>

            <iyy>0.01</iyy>

            <iyz>0</iyz>

            <izz>0.01</izz>

       </inertia>

       <mass>0.845</mass>

  </inertial>

  <collision name='arm\_link\_0\_geom'>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <geometry>

            <mesh>

                 <uri>model://youbot/meshes/arm/arm0\_convex.stl</uri>

            </mesh>

       </geometry>

       <surface>

            <friction>

                 <ode>

                     <mu>0</mu>

                     <mu2>0</mu2>

                     <fdir1>0 0 0</fdir1>

                     <slip1>0</slip1>

                     <slip2>0</slip2>

                 </ode>

                 <torsional>

                     <ode/>

                 </torsional>

            </friction>

            <bounce>

                 <restitution\_coefficient>0</restitution\_coefficient>

                 <threshold>0</threshold>

            </bounce>

            <contact>

                 <ode>

                     <soft\_cfm>0</soft\_cfm>

                     <soft\_erp>0.2</soft\_erp>

                     <kp>1e+13</kp>

                     <kd>1e+11</kd>

                     <max\_vel>-1</max\_vel>

                     <min\_depth>0</min\_depth>

                 </ode>

            </contact>

       </surface>

       <laser\_retro>0</laser\_retro>

       <max\_contacts>10</max\_contacts>

  </collision>

  <visual name='arm\_link\_0\_geom\_visual'>

       <pose frame=''>0 0 0 0 -0 0</pose>

       <geometry>

            <mesh>

                 <uri>model://youbot/meshes/arm/arm0.dae</uri>

            </mesh>

       </geometry>

       <material>

            <script>

                 <uri>model://youbot/materials/scripts/youbot.material</uri>

                 <name>youbot/DarkGrey</name>

            </script>

       </material>

  </visual>

  <gravity>1</gravity>

  <self\_collide>0</self\_collide>

  <kinematic>0</kinematic>

</link>

Joint

Los Joint (Uniones) son uno de los elementos más importantes para definir nuestro movimiento. Estos definen como están unidos jerárquicamente los eslabones y cuál es el tipo de articulación y actuador definidos en la unión.

• Por un lado, tenemos la jerarquía mediante la definición del padre y el hijo. En el caso de nuestra primera unión tiene como padre “base” con el objetivo de fijar el brazo robótico a la base del robot.

• El siguiente punto son los ejes. SDF permite definir 2 ejes de movimiento, aunque en el robot de este proyecto solo definirá un eje por unión. Los ejes podrán moverse de forma rotacional o prismática y permiten definir:

o La dinámica del eje.

o Los límites tanto del ángulo máximo como mínimo que puede adquirir el eje como de la fuerza y velocidad máxima que puede aplicarse.

• Por último, definirá sobre el o los ejes de coordenadas sobre los que puede rotar o desplazarse el eje.

<joint name='arm\_joint\_0' type='revolute'>

  <pose frame=''>0 0 0 0 -0 0</pose>

  <parent>base</parent>

  <child>arm\_link\_0</child>

  <axis>

       <dynamics>

            <damping>0</damping>

            <friction>0</friction>

            <spring\_reference>0</spring\_reference>

            <spring\_stiffness>0</spring\_stiffness>

       </dynamics>

       <limit>

            <lower>0</lower>

            <upper>0</upper>

            <effort>0</effort>

            <velocity>0</velocity>

       </limit>

       <xyz>0 0 1</xyz>

       <use\_parent\_model\_frame>1</use\_parent\_model\_frame>

  </axis>

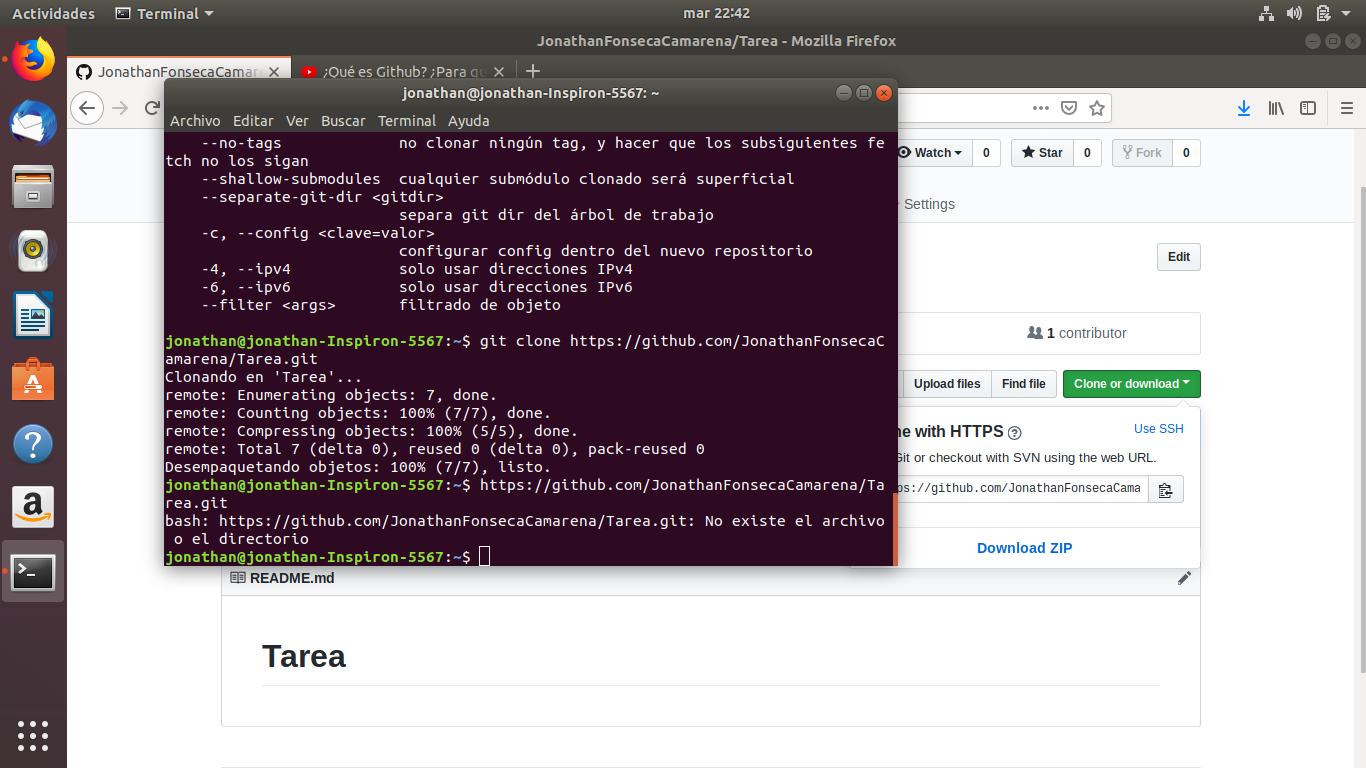
</joint>

Por último, para lanzar el robot utilizaremos el siguiente comando:

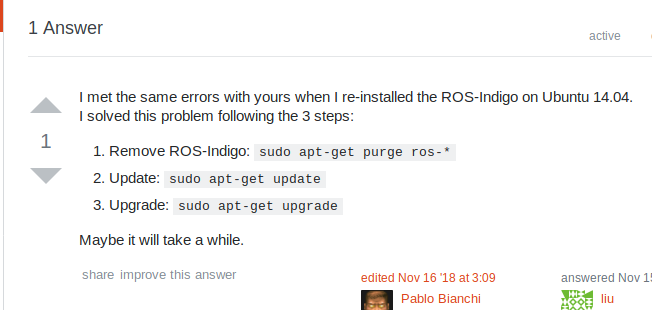
$roslaunch miRobot miRobot.launch

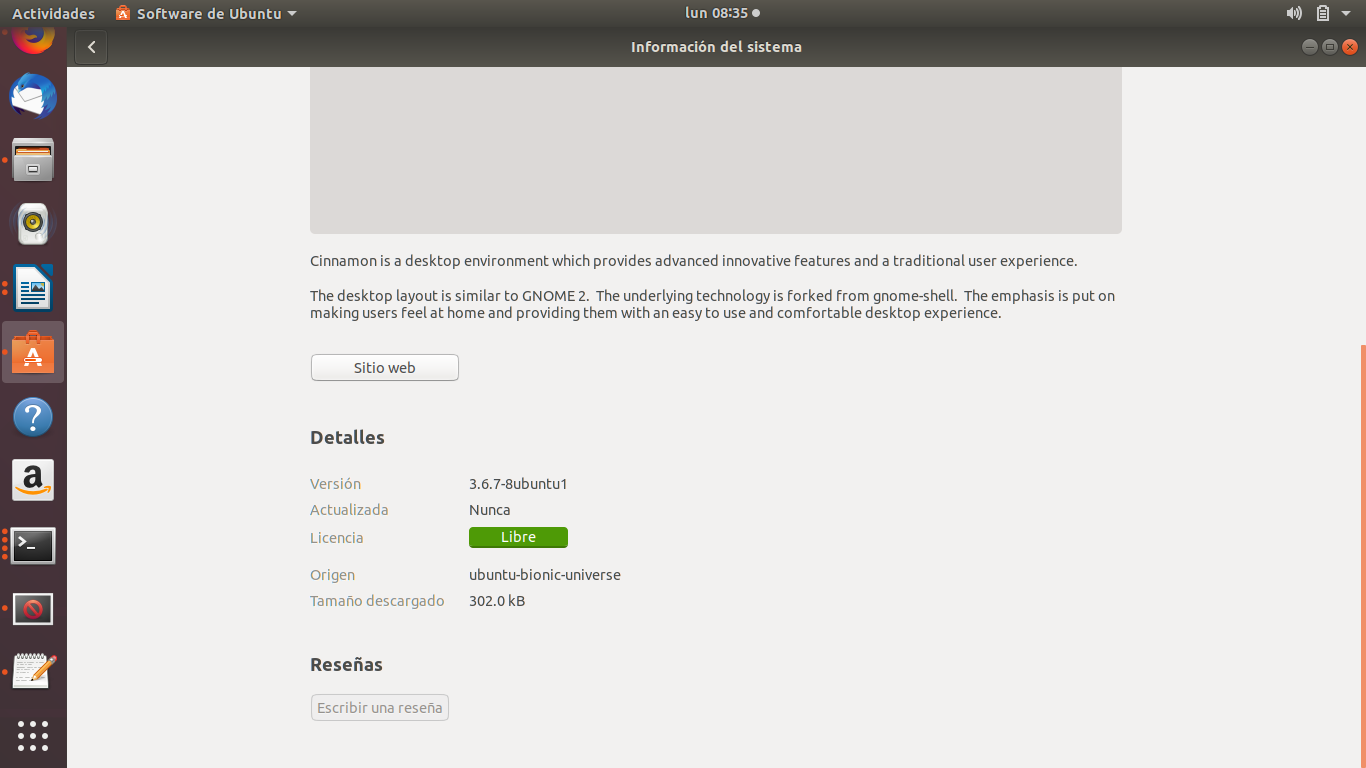
**RESULTADOS:**

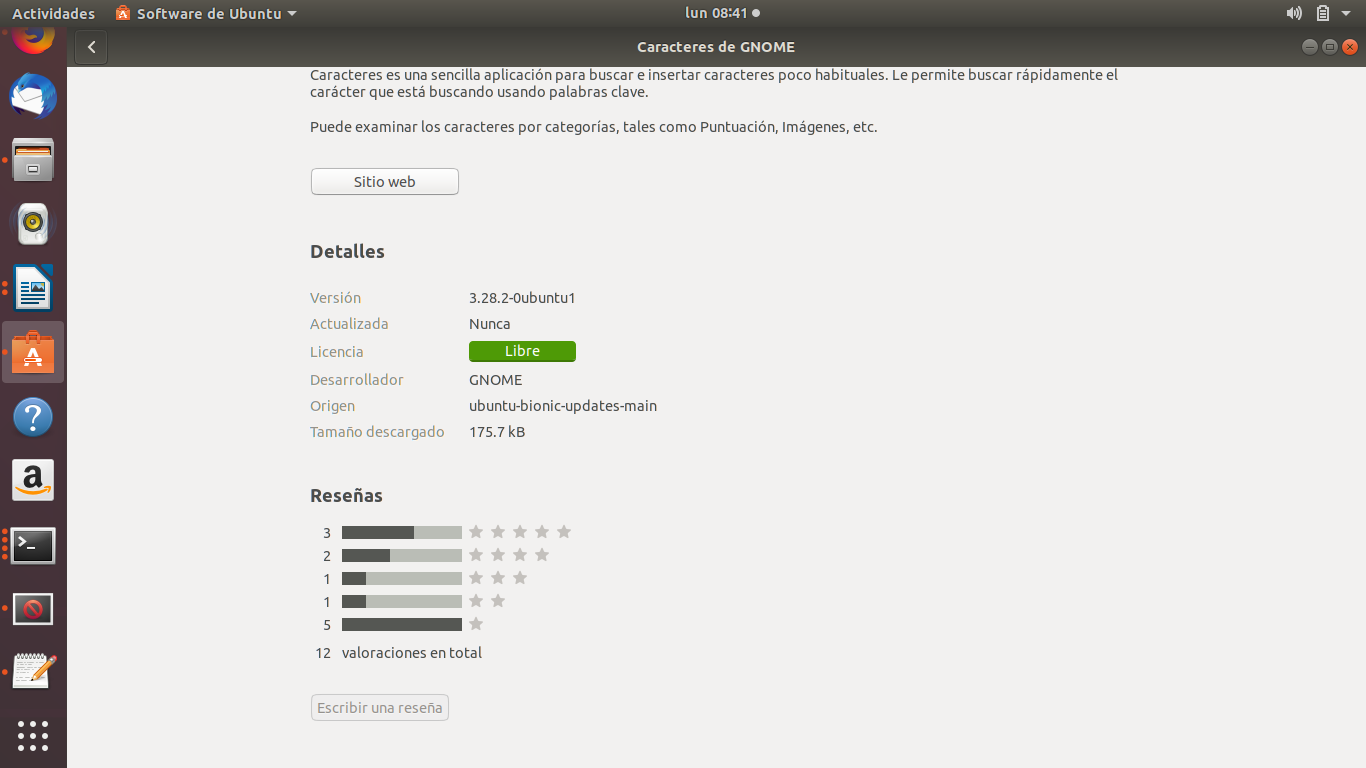
Así empezamos nuestro proyecto de simulación;



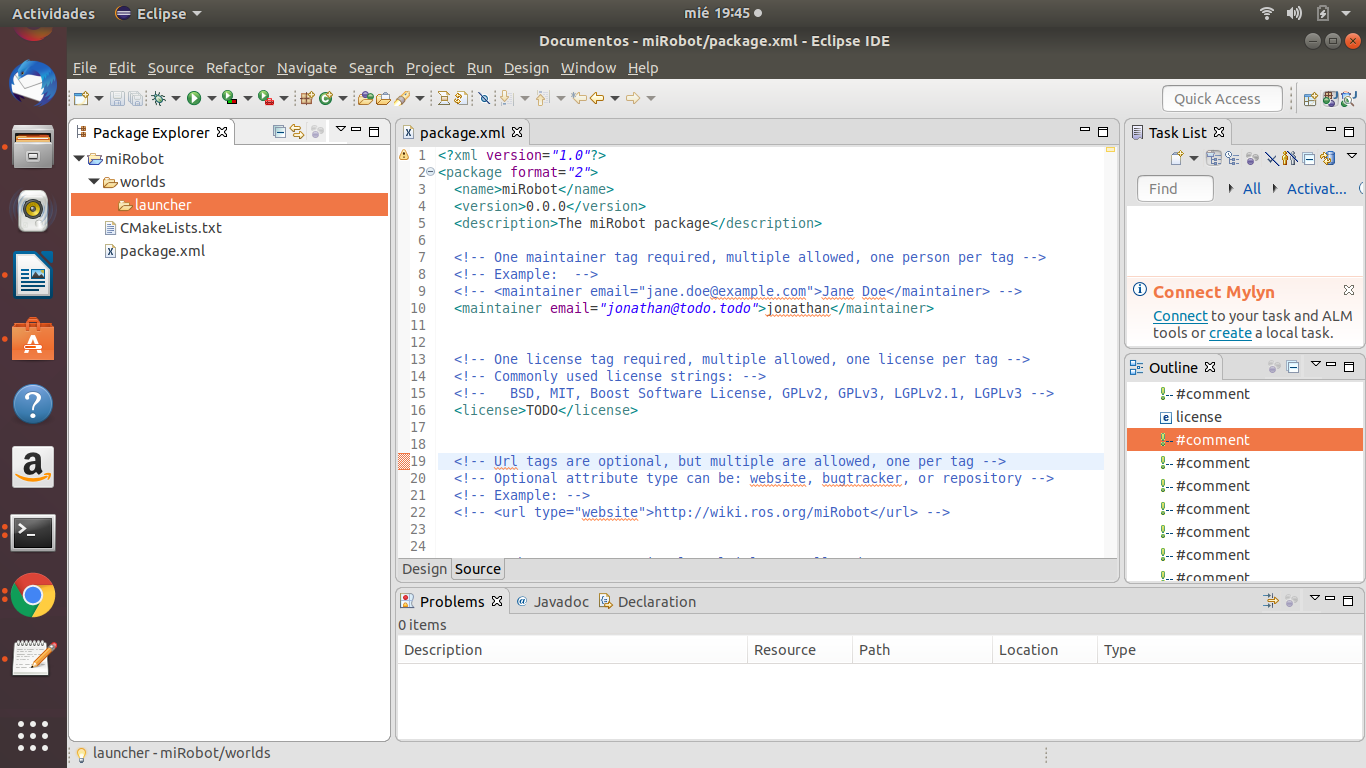
Estas fueron algunas dudas que surgieron en el proceso;

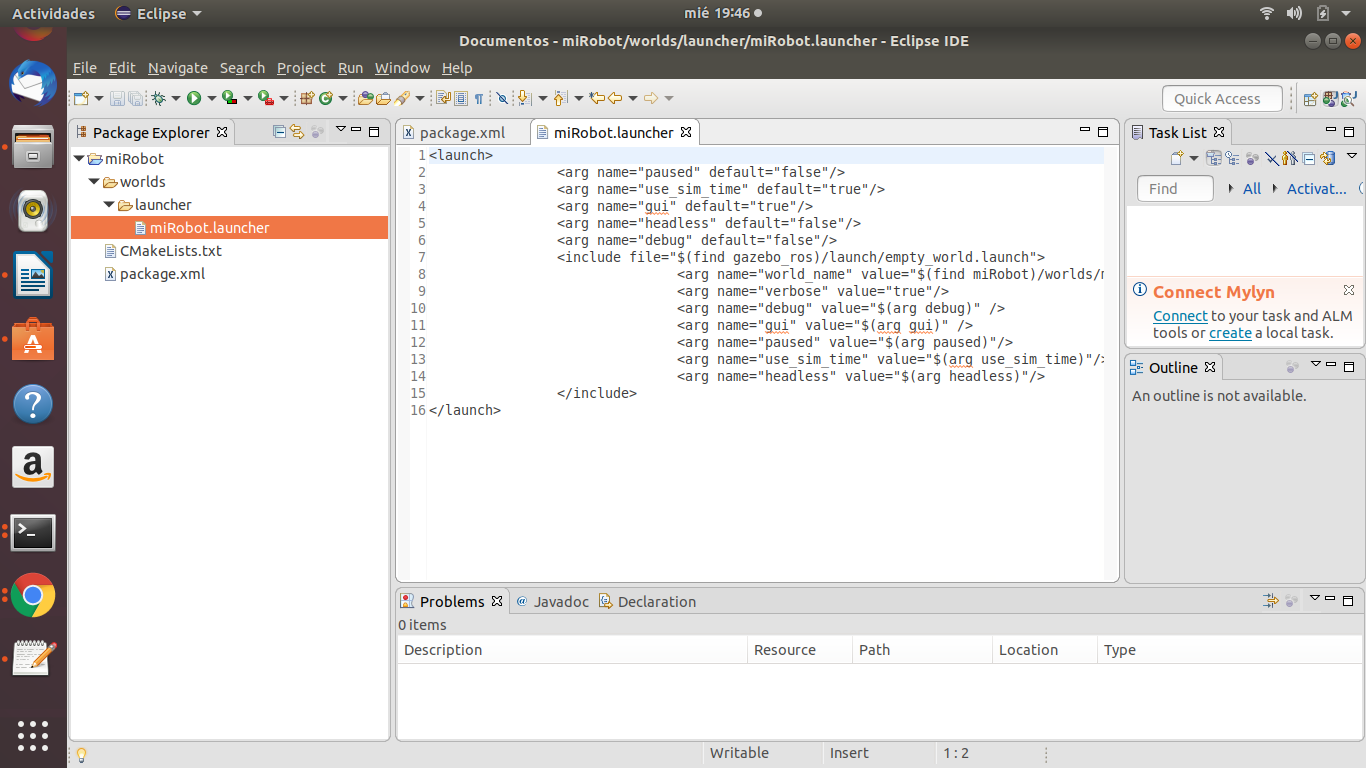


Este es el software requerido para la instalación de los paketes

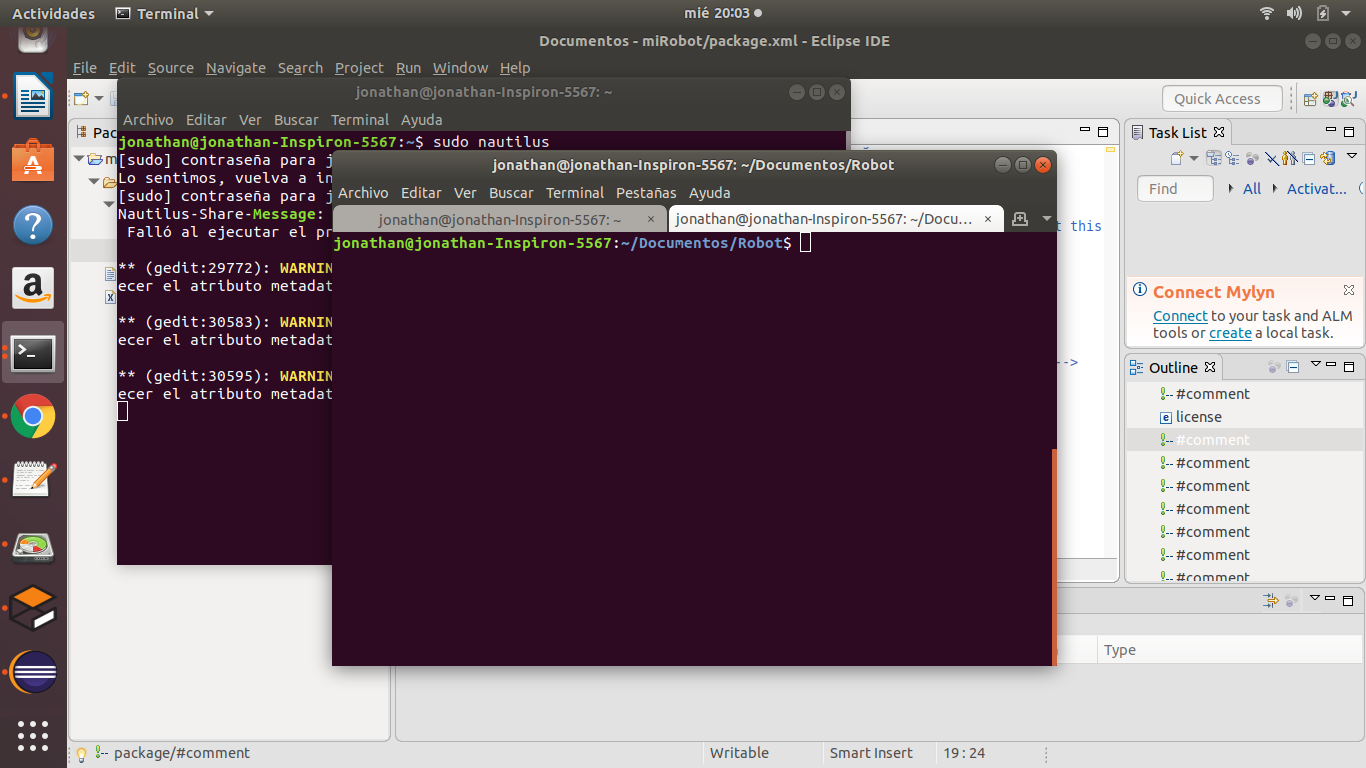


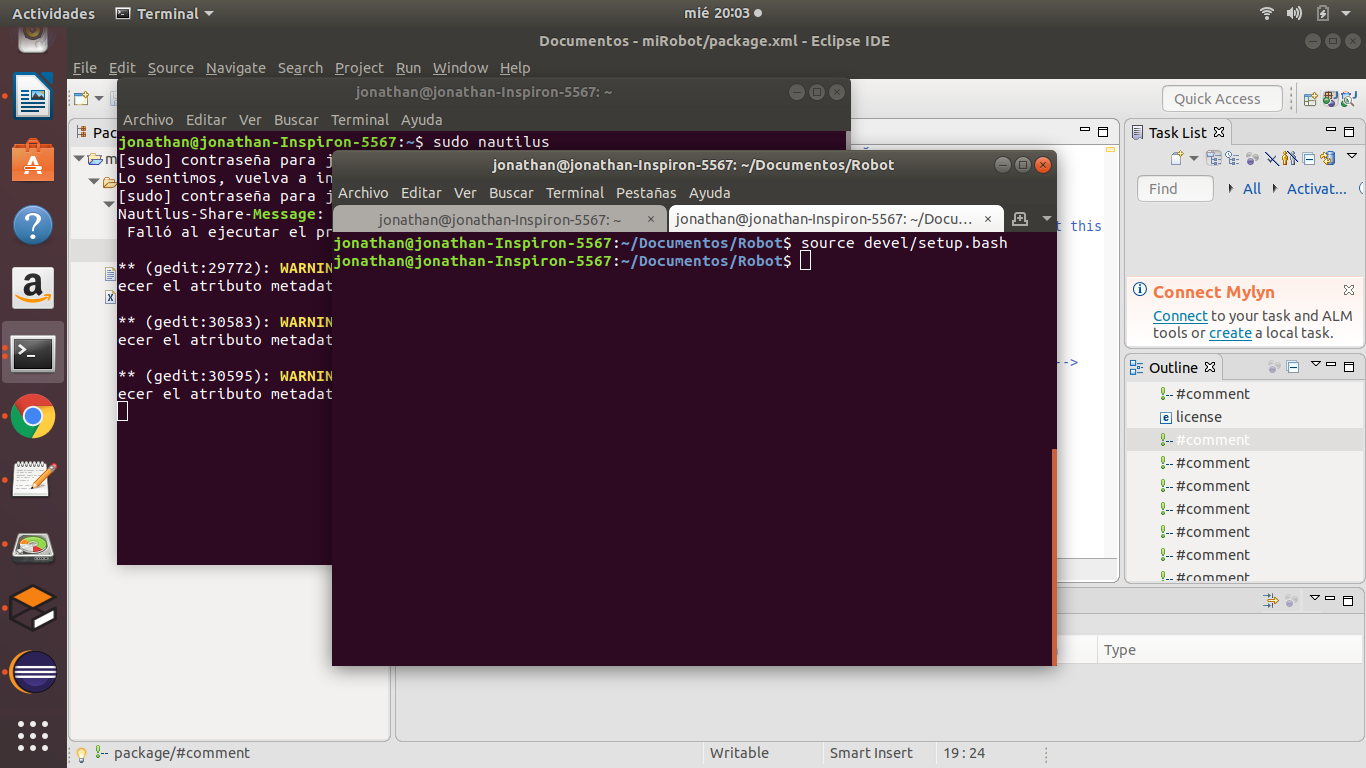
Como vimos anteriormente en el proceso es necesario realizar la ruta del archivo y consigo es necesario crear nuevas carpetas como son las siguientes;

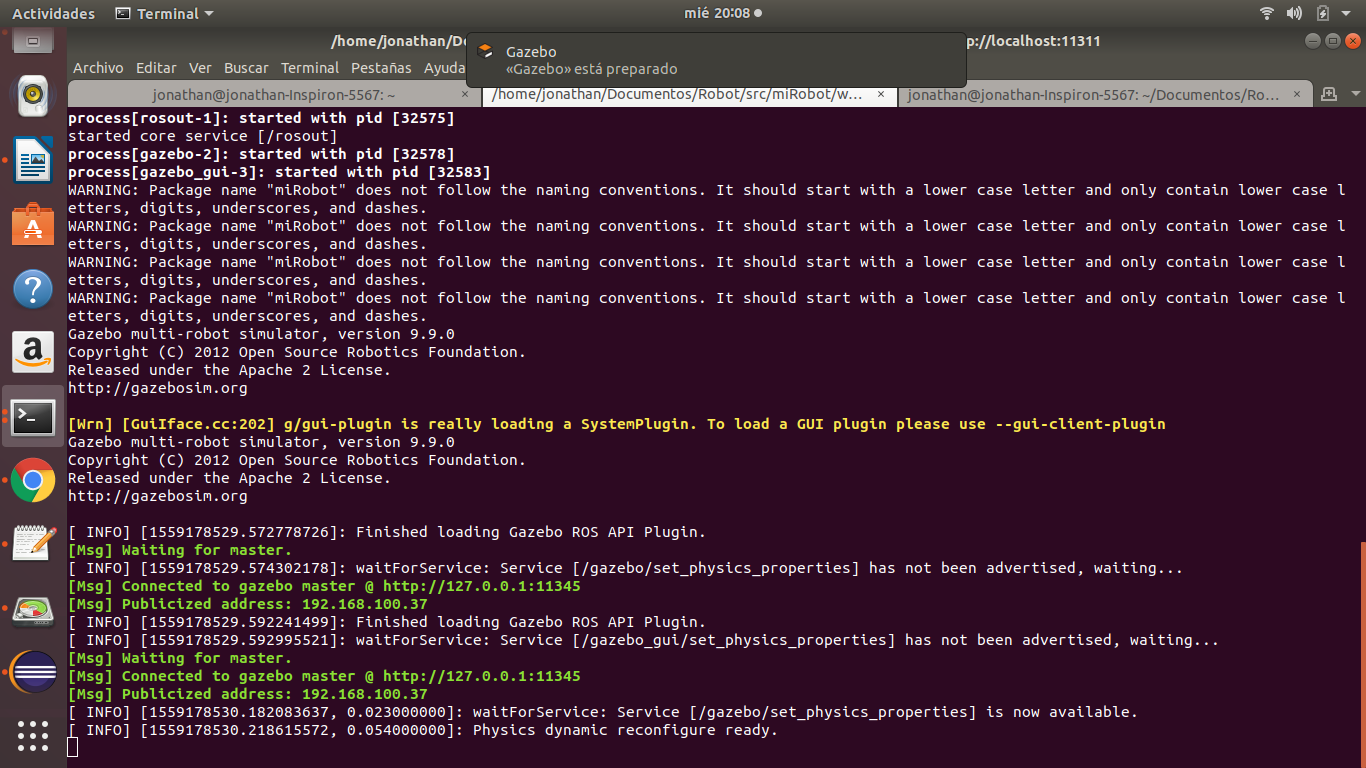




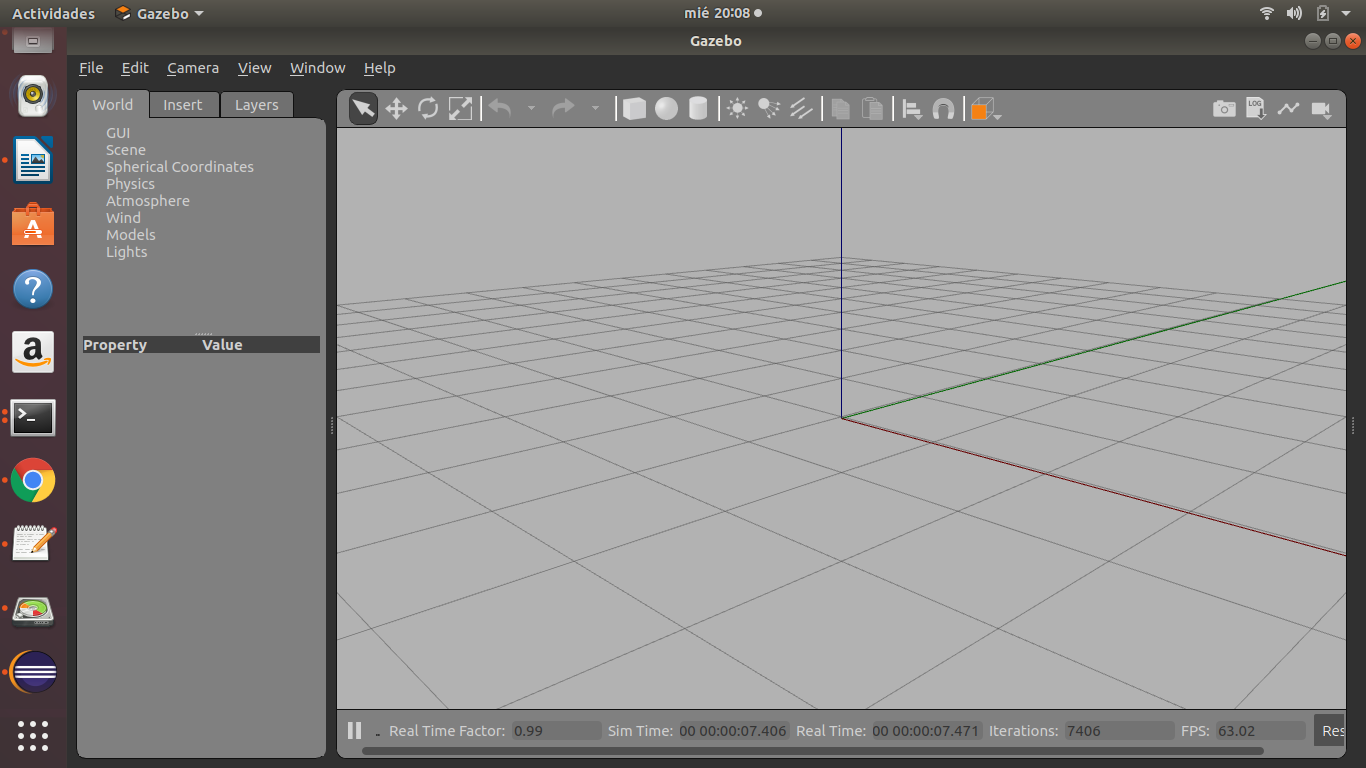
Y ahora en la terminal solo llamaremos a el archivo del titulo como se dice en el procedimiento;



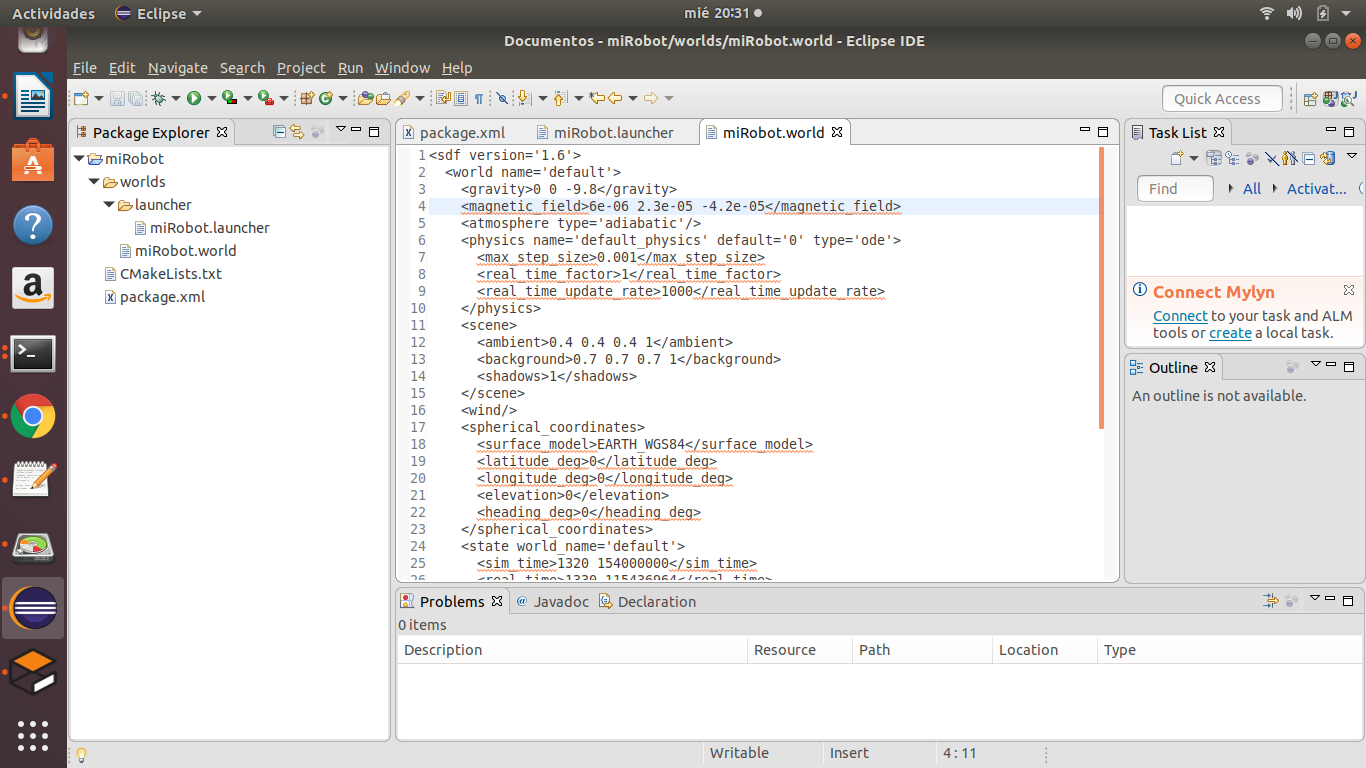




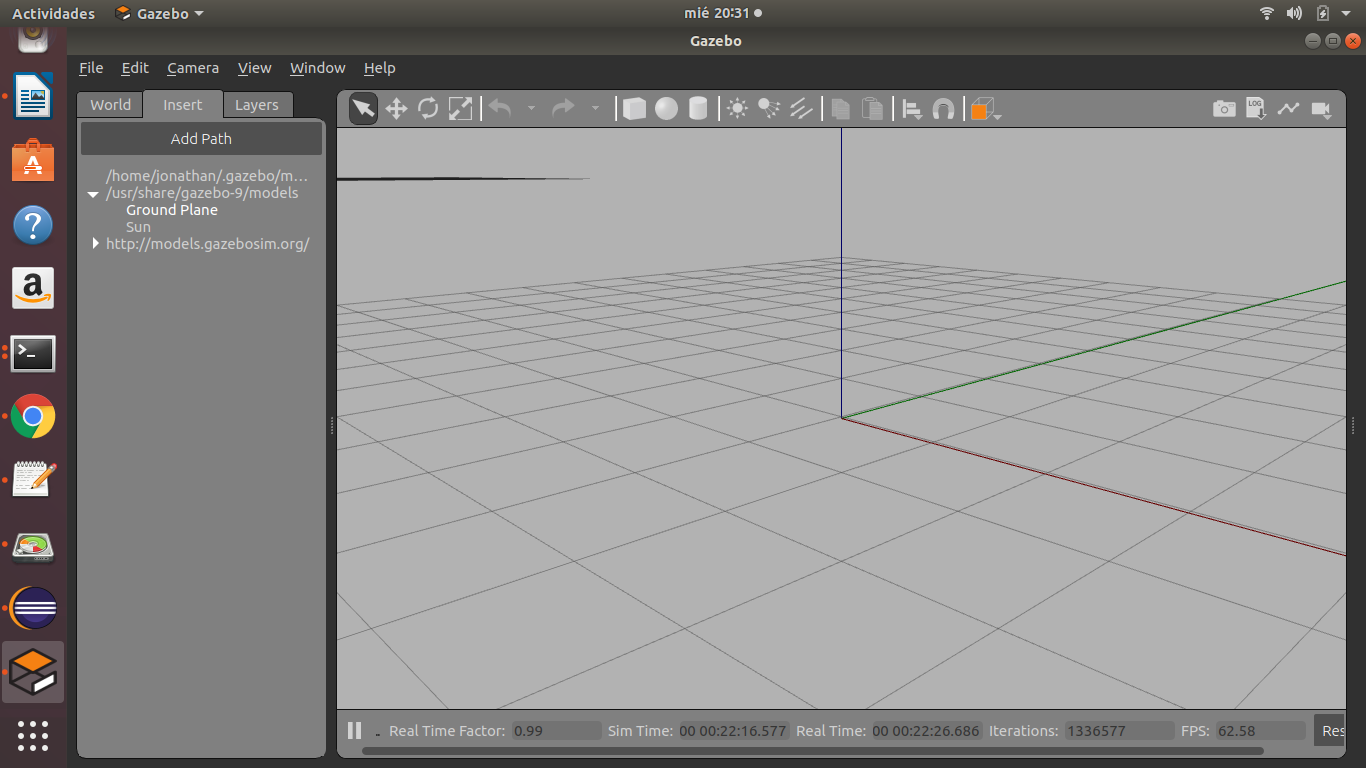
Este será el entono de trabajo ya explicado en el procedimiento;

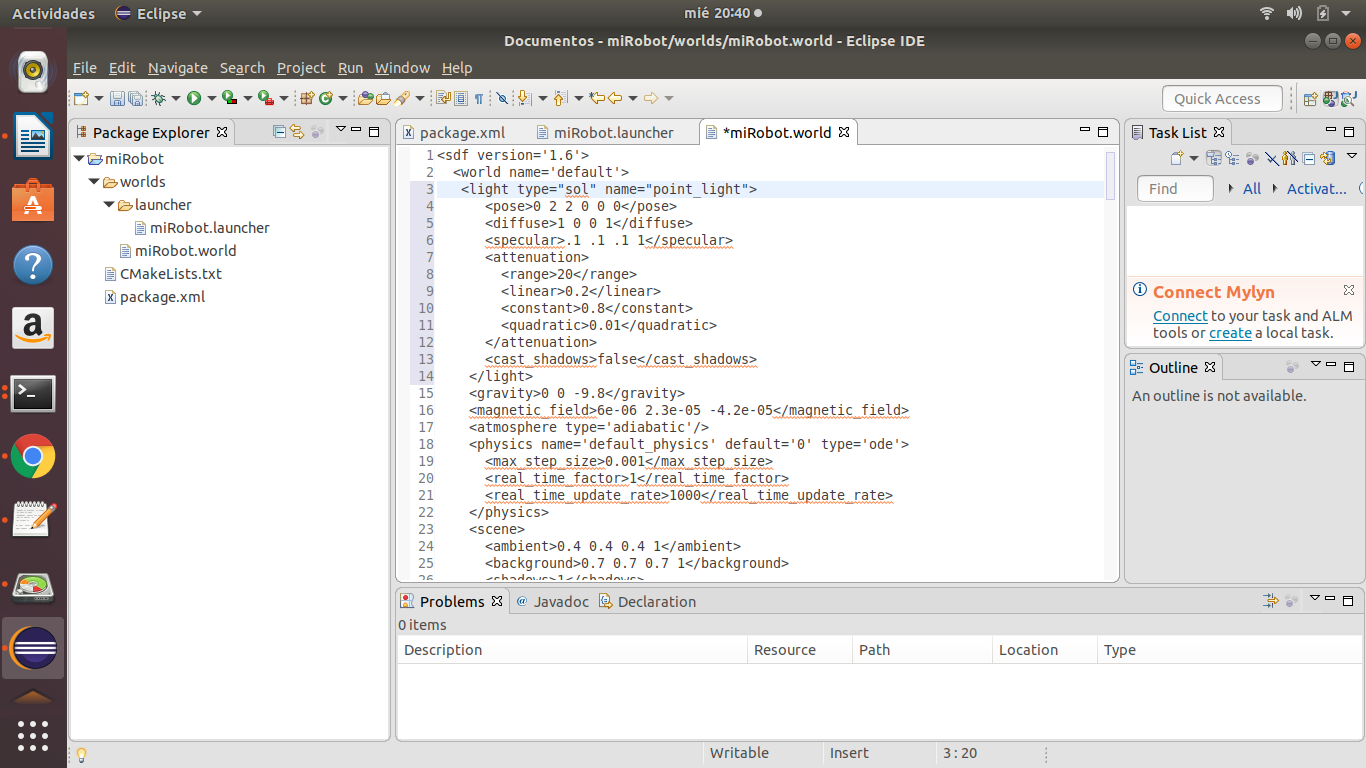


Y esto es parte del código ingresado para la creación del proyecto,

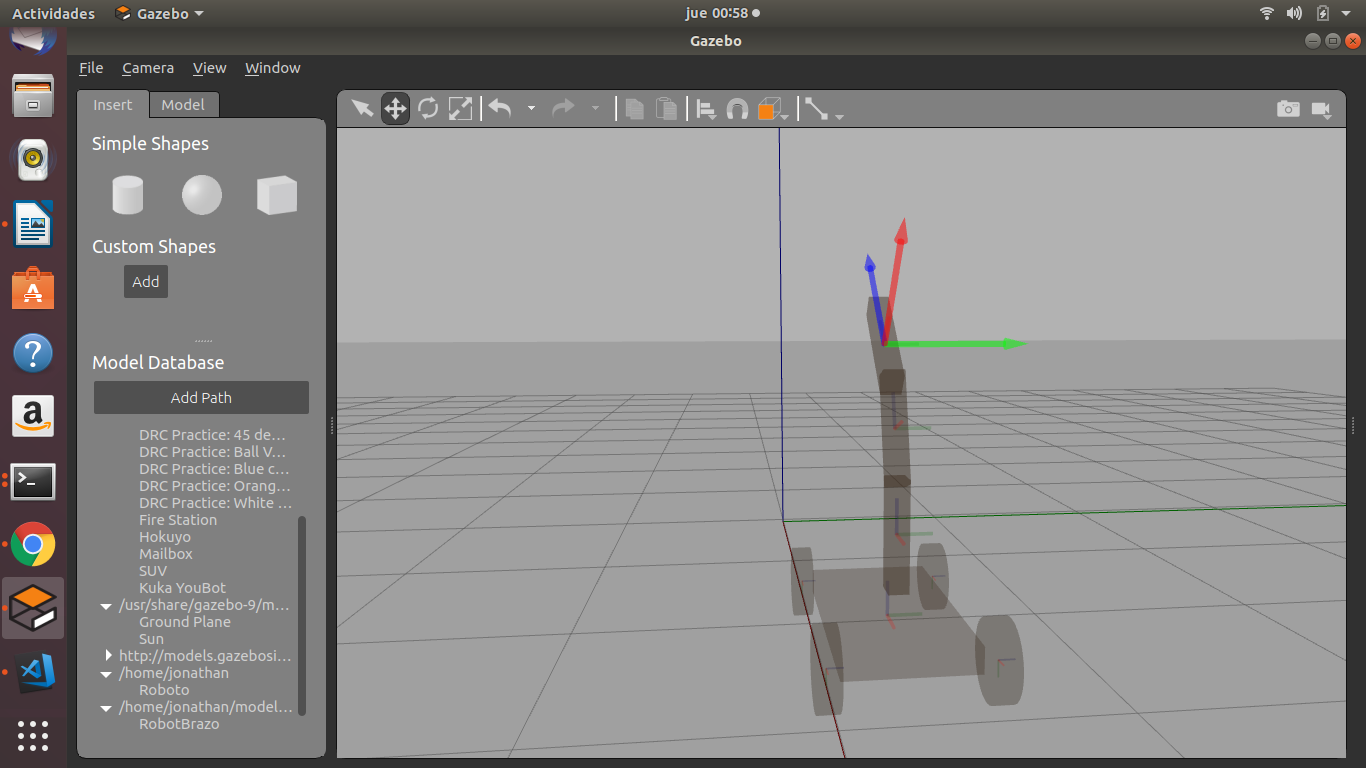


Para poder trabajar en nuestro entorno de trabajo;





Y por último tendremos nuestro “robot” listo para manejarlo y redirigirlo.



**CONCLUSION:**

En esta práctica nos dimos cuenta de cómo simular un “robot”, el entorno de simulación y el desempeño que hacen los comandos en la programación y aprendimos más sobre Gazebo, nos sirvió en saber todo acerca de lo que se necesita desde los diferentes tipos de simuladores hasta todos sus diferentes componentes, así también el diferente tipo de características de movimiento, lo que queremos decir es que aprendimos un poco más sobre la morfología en general y así nos fue más fácil el manejo de el robot.