Robot Automático de logística - RAL

Esteban Ramos*, Juan Camilo Gutierrez[1], Nelson Delgado[2]

En los últimos años gracias al aumento de las compras electrónicas se ha aumentado el trabajo que deben realizar los empleados de los centros de logística, aumentando la carga física que deben soportar a transportar las mercancías las cuales algunas son muy pesadas, esto aumenta los tiempos de ejecución para la organización de los almacenes y puede llegar a causar lesiones físicas a los empleados. Nuestro proyecto consta de un sistema de robots autónomos los cuales se encargan del transporte y organización de las mercancías dentro de estos centros de logística. Nuestros robots serán más económicos que la actual competencia, permitiendo levantar cargas sin que el dispositivo pierda su estabilidad. Con nuestros robots de carga autónomos nos permitirán llevar mercancías de una manera rápida y segura dentro del bodegaje, sin poner en riesgo la seguridad del centro y sus empleados. Las cargas que podrá levantar el robot serán de hasta 250 kilogramos. La mercancía estará ubicada en estanterías las cuales estarán demarcadas para que el robot las pueda identificar. El robot gracias a su tecnología de ubicación espacial podrá saber a qué lugar debe llevar la mercancía que será pedida por un operario. Nuestro sistema de gestión de logística permitirá la automatización de procesos que llevan muchas veces horas o días cuando son realizadas manualmente, entre estos procesos se encuentra la localización de la mercancía y su transporte.

INTRODUCTION

Los robots móviles que utilizan ruedas son vehículos cuyo movimiento es el resultado de la interacción de sus ruedas con el suelo. Uno de los principales problemas relacionados con estos robots es el fenómeno de deslizamiento. Este es un problema muy conocido en el campo de la automoción, que en el caso de estos robots a menudo es descuidado. Además, las ruedas de los robots suelen tener una geometría y propiedades diferentes a la de los automóviles. Este tipo de ruedas normalmente utilizan neumáticos no neumáticos, por lo cual exigen una investigación específica. Los robots móviles se denominan según la distribución de sus ruedas y sus ángulos de libertad de movimiento.

Entre los robots móviles podemos encontrar los vehículos AGV (Automated Guided Vehicle) los cuales son utilizados en la industria principalmente para el transporte de materiales. El principal mercado consumidor de los robots AGV es la industria automotriz, aunque también es muy conocido en otras industrias como lo es la de logística en almacenes y centros de distribución. Gracias a este tipo de robots podemos notar un aumento en productividad, una mayor flexibilidad en los tiempos de entrenamiento, una mejor utilización del espacio, poder garantizar la seguridad de los empleados y reducir los costes de operación en general [1]. Un claro ejemplo de este tipo de robots es el robot kiva de Amazon el cual se encarga de la logística en muchos de sus almacenes, agilizando el tiempo de entrega por medio del aumento de agilidad a la hora de localizar los productos del almacén adquiridos por las personas en sus tiendas virtuales y llevarlos a un operario que se encarga de agrupar los diferentes artículos que hacen parte del pedido realizado dando inicio al proceso de embalaje.

PROBLEMÁTICA

El panorama mundial nos ha demostrado el aprovechamiento de técnicas mecatrónicas con tecnologías para la gestión de almacenes de forma automática e inteligente. Entre los múltiples ejemplos sobre este aprovechamiento la multinacional amazon utiliza robots encargados del transporte y gestión de productos en sus bodegas, estos robots cuentan con tecnología punta lo cual les permite obtener resultados "premium" en la gestión de sus productos. A nivel nacional son pocas las empresas que están interesadas en adquirir tecnologías que serían beneficiosas para el manejo de productos con alto flujo. Este poco interés es debido a la gran inversión para la compra, implementación y mantenimiento de estas tecnologías normalmente extranjeras.

MOTIVACIÓN

Contribuir en el desarrollo y prototipo para la creación de un robot inteligente con la capacidad de encargar, recoger y llevar objetos de un destino a otro utilizando técnicas de robótica, poniendo este proyecto a la disposición de más personas ampliando la asequibilidad de estas tecnologías, también proporcionando un menor tiempo de operación al igual de la de la carga física de los operarios de los centros de logística de manera que cree valor agregado para las diferentes compañías de logísticas y a la vez usar este prototipo para la investigación por futuros estudiantes y curiosos por la mecatrónica.

IMPACTOS RELEVANTES

En la actualidad la optimización industrial es un campo fundamental para mejorar el desempeño de los procesos industriales logísticos y de fabricación, ya que se requieren avances tecnológicos para satisfacer las necesidades de la industria actual, es por ello que nuestro proyecto busca cumplir con los parámetros correspondientes en el campo del transporte y

distribución. en el área industrial, que son fáciles de manejar, transportan cargas pesadas, de tamaño compacto.

SOLUCION PROPUESTA

Nuestra solución es un robot con tracción 4wd el cual cuenta con 4 llantas motorizadas con movimiento. El robot cuenta con un sistema de tijeras de elevación para el sistema de carga.

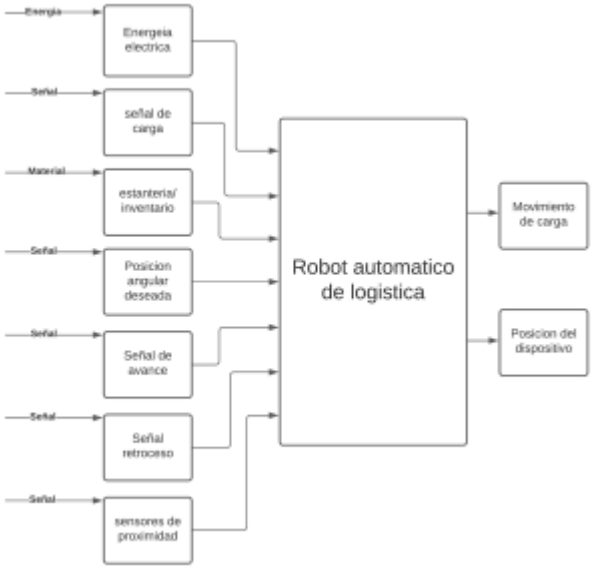


Fig.1 Caja negra

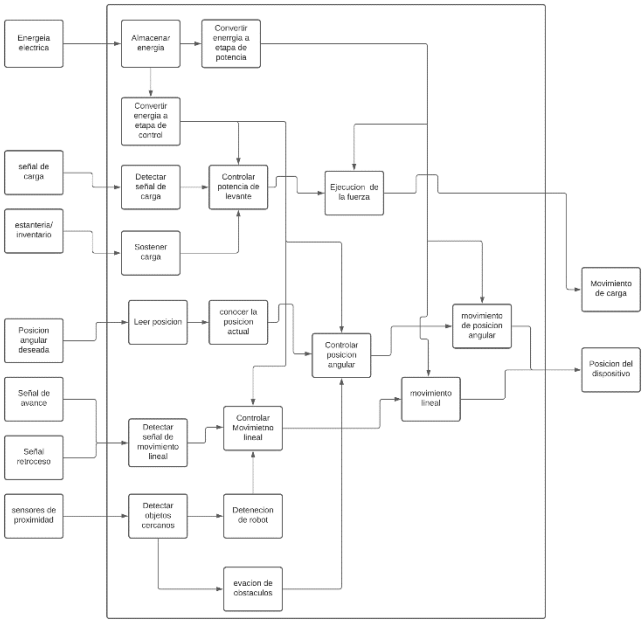


Fig.2 Caja transparente

CINEMÁTICA DEL ROBOT

La estructura cinemática del robot [2]

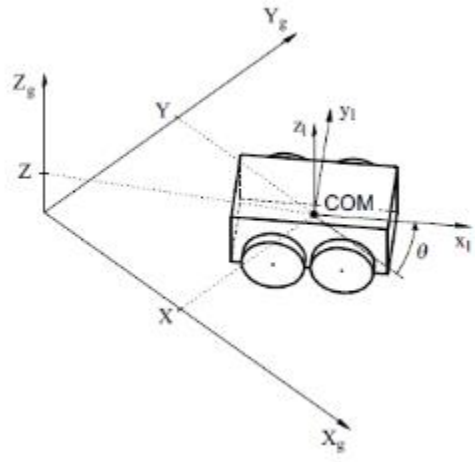


Fig.3

Primero se asigna un marco de referencia local dentro del robot que se denota por (Xl, Yl, Zl) como se observa en la imagen anterior el centro de masa en el marco inercial COM = (X, Y, Z) considerando que el robot realiza sus movimientos en el plano.

se puede suponer que el robot se mueva a una velocidad lineal expresada en las coordenadas locales como $V = [Vx, Vy, 0]^T$, y rota con una velocidad angular $w = [0,0,w]^T$.

$q = [X\ Y\ \theta]^T$ describe la posición del robot, $\dot{q} = [\dot{X}\ \dot{Y}\ \dot{\theta}]^T$ Es el vector de velocidades del robot.

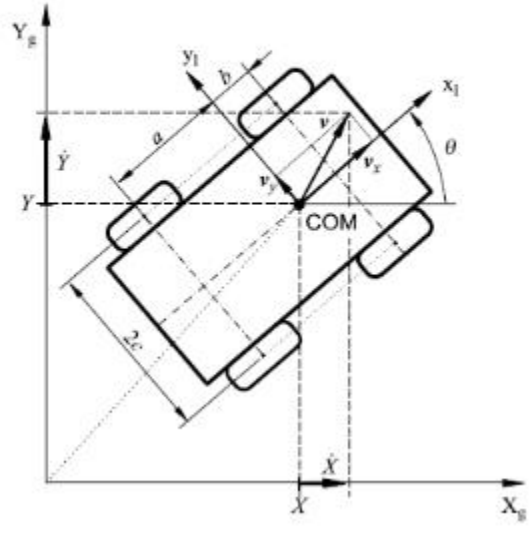


Fig.4

X' e Y' están relacionadas a las coordenadas locales de velocidad de la siguiente manera:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{bmatrix} \begin{bmatrix} Vx \\ Vy \end{bmatrix}$$

El movimiento en el plano también lo podemos representar como θ´ = w.

Vx = ri wi

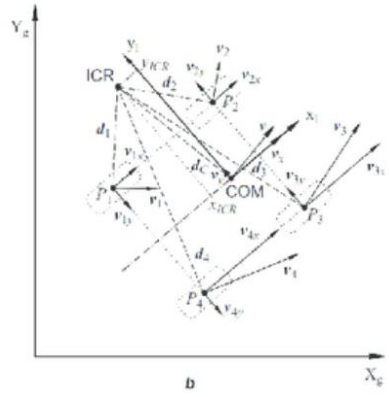


Fig.4

Las distancias d1, d2, d3, d4 se miden desde el punto de contacto de las ruedas a el centro de rotación. Se puede determinar que:

d1y = d2y
d3y = d4y
d1x = d4x
d2x = d3x

Gracias a esto podemos decir que las fórmulas de las velocidades serán:

VL= V1x= V2x
VR = V3x = V4X
VF = V2y = V3y
VB = V1y = V4y

Gracias a las fórmulas anteriores podemos llegar a la relación presentada a continuación:

$$n = \begin{bmatrix} Vx \\ W \end{bmatrix} = \begin{bmatrix} \frac{W_L + W_R}{2} \\ \frac{-W_L + W_R}{2C} \end{bmatrix}$$

donde n es la variable cinemática de control y para la velocidad tenemos que:
q' = S(q)n

$$S(q) = \begin{bmatrix} cos\theta & X_{IGR}\, sin\theta \\ sin\theta & -X_{IGR} cos\theta \\ 0 & 1 \end{bmatrix}$$

En la ecuación anterior se puede percibir que XIGR afecta directamente en el movimiento del robot. dado que XIGR está dado por la distancia entre los ejes, el robot se puede deslizar y perder estabilidad. Este problema anteriormente descrito puede ser solucionado por:
Vx = X0 * θ'
donde X0 da una coordenada fija del ICR
X0 ∈ (-a,b)
por último, tenemos que

$$S(q) = \begin{bmatrix} cos\theta & X_0\, sin\theta \\ sin\theta & -X_0 cos\theta \\ 0 & 1 \end{bmatrix}$$

X0 es un valor arbitrario. al escogerlo como cero hace que su modelo cinemático se aproxime al de un robot diferencial de dos ruedas.

$$X(t) = X(0) + \int_0^t r \frac{w_R + w_L}{2} \cos\theta \, dt$$

$$Y(t) = Y(0) + \int_0^t r \frac{w_L + w_R}{2} \sin\theta \, dt$$

$$\theta(t) = \theta(0) + \int_0^t r \frac{w_R - w_L}{2C} \, dt$$

Las entradas del modelo cinemático son las velocidades de las ruedas respectivamente.

SIMULACIÓN DEL PROBLEMA Y DE LA SOLUCIÓN USANDO GAZEBO

En la simulación de gazebo se puede observar en primer lugar el robot descrito anteriormente y en segundo lugar podemos observar la problemática que serian las estanterías ahora bien es relevante visualizar la forma en la que trabaja el robot en la siguiente secuencia de imágenes se observa el robot en diferentes angulos y realizando la solución planteada con el sistema de levante por medio de un movimiento prismático ver fig.8 y 9 (se observa que levanta la estantaria una distancia del piso necesaria para trasladar la estanteria).

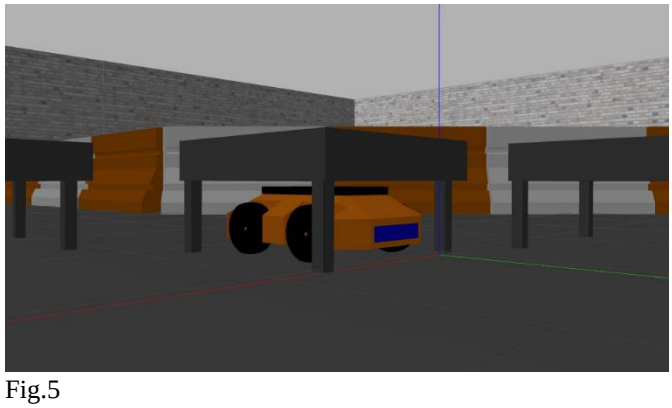


Fig.5

Fig.6

Fig.7

Fig.8

Fig.9

VISUALIZACIÓN DEL PROBLEMA Y DE LA SOLUCIÓN USANDO RVIZ

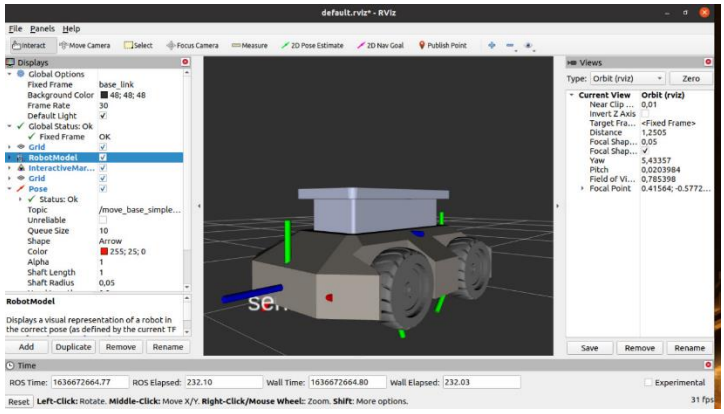


Fig.10

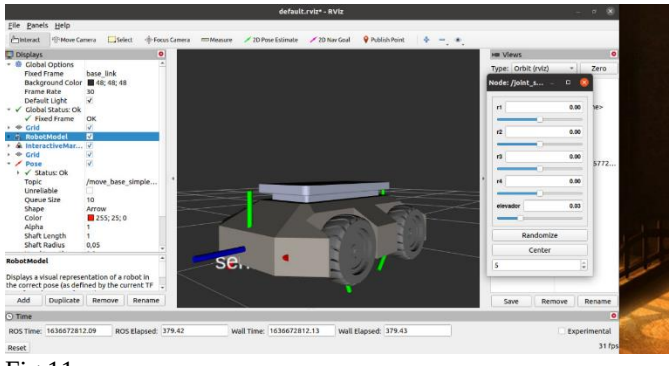


Fig.11

TRABAJO INDEPENDIENTE

Para la realización de este proyecto nosotros como grupo nos hemos tomado la tarea en investigar sobre la cinemática de los robots móviles debido a que estos se mueven mediante sus llantas además estas son las que le proporcionan las restricciones de movimiento a nuestro robot debido a que son el medio por el cual nuestro robot podrá moverse por su propia cuenta.

En internet nos hemos encontrado con mucha información para el entendimiento de nuestra propuesta de proyecto en el curso de robótica de la universidad autónoma de occidente debido a que el motivo de este trabajo fue aplicar los conocimientos aprendidos durante el transcurso del curso además de los tutoriales propuestos por nuestro profesor.

Los tutoriales de The Construc fueron muy útiles para el entendimiento, pero debido algunos de sus cursos en su pagina web eran de pago por lo que hemos optado por buscar videos y paginas en otros idiomas para poder encontrar información útil para la realización de nuestro proyecto.

Para empezar nuestra búsqueda hemos dividido por 8 etapas de investigación las cuales se podrán ver reflejadas en la siguiente lista.

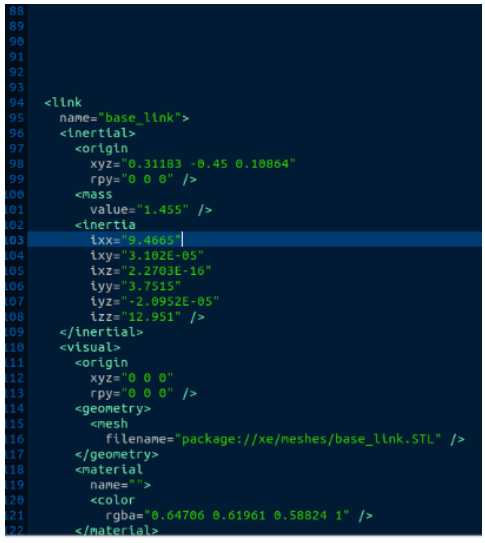
1) Ubuntu 20.04 y sus herramientas.
2) Creación de los archivos .urdf y su utilidad.
3) Cinemática directa de robots móviles.
4) Links y articulaciones para robots móviles.
5) Tele-operación de Robots Moviles.
6) Plug-ins y Controladores en Gazebo.
7) Creación de Mundos en Gazebo.
8) Definir Objetos Externos por archivos STL.

1) Para entender el funcionamiento y las herramientas útiles para nuestros robots debido a que es necesario para los ingenieros poder visualizar el funcionamiento del robot en un ambiente similar al real para poder tener aproximaciones y resultados similares para cuando nuestro robot se encuentre en operación.
Ubuntu 20.04 tiene ROS que es un sistema operativo de código abierto que posee varias librerías y herramientas para ayudar a los desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos y diferentes herramientas de simulación y comunicación por mensajes. En este caso hemos utilizado Ubuntu 20.04 debido a que en internet se comenta que es una de las versiones más completas de Ubuntu y sin errores.

Ubuntu 20.04 tiene el simulador 3D Gazebo que posibilita el comportamiento de un robot en un mundo virtual, además permite otras opciones para diseñar robots de forma personalizada, crear mundos personalizados utilizando CAD e importar modelos ya creados.

Fig.12 Icono Gazebo

Por lo que investigando profundamente sobre los archivos que admite Gazebo para simular nuestros robots hemos encontrado que admite archivos .URDF y .XACRO donde se menciona que los archivos .URDF es una extensión de archivo que posee la información, geometría y apariencia de nuestro robot debido a que se usa como un archivo de descripción de robot para simulación. Solidworks tiene una extensión donde permite crear un modelado en 3D del robot y luego convertirlo en formato .URDF para poder ser simulado en Gazebo.



```
<link
  name="base_link">
  <inertial>
    <origin
      xyz="0.31183 -0.45 0.10864"
      rpy="0 0 0" />
    <mass
      value="1.455" />
    <inertia
      ixx="9.4665"
      ixy="3.102E-05"
      ixz="2.2703E-16"
      iyy="3.7515"
      iyz="-2.0952E-05"
      izz="12.951" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://xe/meshes/base_link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.64706 0.61961 0.58824 1" />
    </material>
```

Fig.13 Imagen Fuente Propia de nuestro Robot.

También esta RVIZ que es una herramienta de visualización en 3D para aplicaciones de ROS. Proporciona una vista del modelo del robot, captura la información de los sensores del robot y reproduce los datos capturados por el mismo además de mostrar los datos captados por la cámara o los láseres.

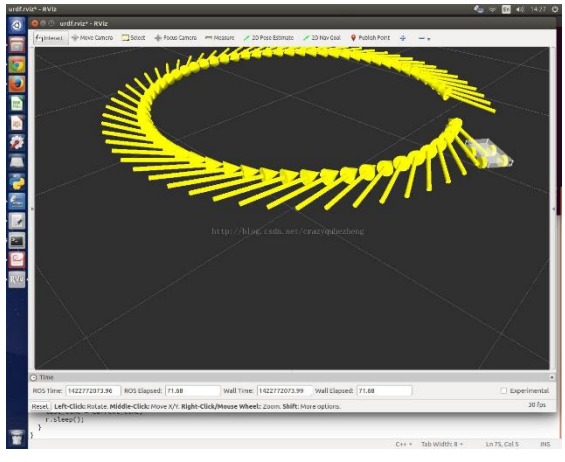


Fig.14
Imagen tomada de: https://answers.ros.org/question/223695/tf-origin-is-away-from-the-base_link/

2) Para la creación de los archivos .URDF y entender su utilidad fue muy valiosa la información proporcionada por nuestro profesor durante el transcurso del curso debido a que tuvimos clases enfocadas en entender la importancia de los archivos .URDF mas sin embargo nuestro grupo también se dio la tarea de buscar información en internet para obtener una idea más sólida sobre este tipo de archivos y al mismo tiempo de su creación.

La importancia de este tipo de archivos es de que podremos ver la estructura la cual está constituido un robot debido a que estos se

componen entre piezas base, hijas, links, joints, sensores y efectores finales para el funcionamiento correcto de los robots. En ROS y con ayuda de Linux en Ubuntu 20.04 tenemos la posibilidad de descubrir de que se componen nuestros archivos .URDF para crear un PDF donde se nos suministre la información necesaria de este por medio de la descripción de sus partes por medio del siguiente diagrama:

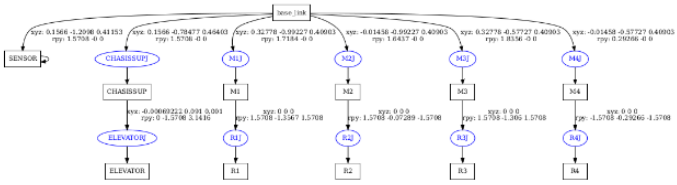


Fig.15

Por lo que es una buena herramienta para poder saber el tipo de robot que contiene nuestro archivo .URDF para así clasificarlo y tener una referencia de su constitución y funcionamiento.

3) Para entender el funcionamiento y la cinemática de nuestro robot miramos el siguiente video.

Fig.16
Imagen tomada de: https://www.youtube.com/watch?v=hKTbdZHvK_A

Como se puede apreciar en la imagen el robot está compuesto de 4 llantas fijas motorizadas que hacen posible el desplazamiento de nuestro robot. Pensando en que nuestro robot debe tener la capacidad de llevar carga de un lugar A a un lugar B pensamos en el diseño de nuestro robot de la manera en la cual se está mostrando el video.

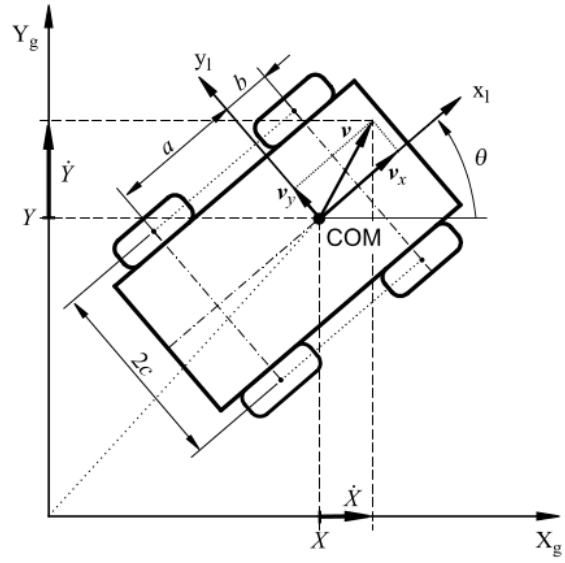


Fig.17
Imagen tomada de: http://matwbn.icm.edu.pl/ksiazki/amc/amc14/amc1445.pdf

También para el entendimiento de los modelos cinemáticos de los robots móviles nos hemos apoyado del siguiente video el cual nos explica el modelo cinemático de un robot móvil diferencial para poder

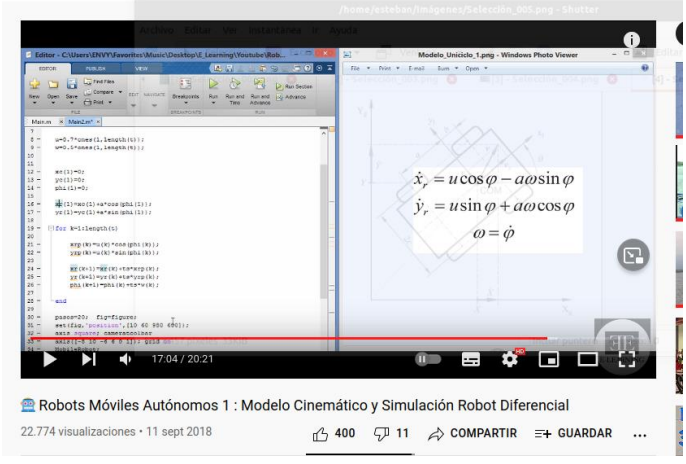


Fig.18

4) Para entender la estructura de los robots móviles nos hemos basado en el siguiente enlace: http://gazebosim.org/tutorials/?tut=build_robot

En este tutorial se nos indica cómo crear correctamente el espacio de trabajo donde irán los archivos y configuraciones necesarias para la simulación de nuestro robot.
Según el tutorial se nos indica la estructura básica de un robot es la siguiente:

```
<?xml version='1.0'?>
<sdf version='1.4'>
  <model name="my_robot">
    <static>true</static>
    <link name='chassis'>
      <pose>0 0 .1 0 0 0</pose>
      <collision name='collision'>
        <geometry>
          <box>
            <size>.4 .2 .1</size>
          </box>
        </geometry>
      </collision>

      <visual name='visual'>
        <geometry>
          <box>
            <size>.4 .2 .1</size>
          </box>
        </geometry>
      </visual>
```

Fig.19
Imagen toma de del tutorial ROS http://gazebosim.org/tutorials/?tut=build_robot

Siguiendo las instrucciones que se nos indican en el tutorial se nos fue posible realizar y entender los archivos que maneja Gazebo y Rviz para la simulación de robots.

5) Para la tele-operación de nuestro robot móvil de 4 llantas hemos utilizado el plugin que nos explico nuestro profesor en clase que consiste en lo siguiente.

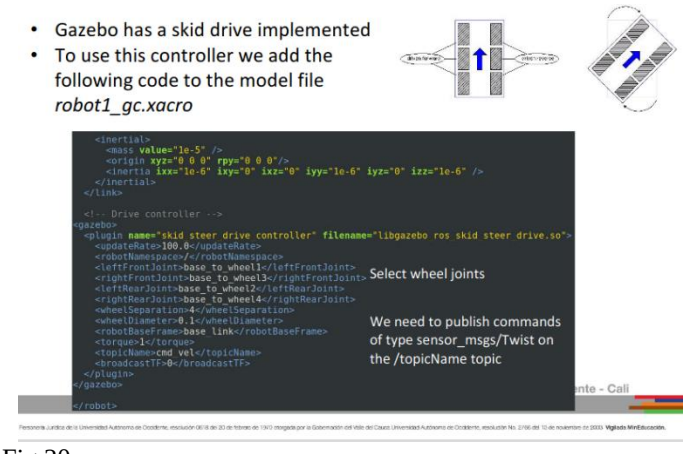


Fig.20
Imagen tomada de: https://campus.uaovirtual.edu.co/pluginfile.php/290221/mod_resource/content/1/RS10a%20-%20Robot%20simulation%20with%20Gazebo%20II.pdf

Durante el transcurso del curso de robótica en la universidad autónoma de occidente nuestro profesor nos comentó el funcionamiento del Plug-in para la tele-operación de un robot móvil de 4 llantas por medio del teclado mediante el envío de mensajes en el ros-topic de CMD/VEL que este sirve para controlar las direcciones X, Y, Z de nuestro robot.
Para entender mejor le funcionamiento de este Plug-in fue necesario revisar las funciones de los diferentes ros-topic message que nos sirven para controlar diferentes parámetros de simulación en nuestro robot para controlarlo.
En la página de ROS.wiki: http://wiki.ros.org/rostopic se nos explican los diferentes rostopic que existen en el entorno de gazebo. Rostopic contiene la herramienta de línea de comandos que nos dan información de depuración en ros. Con esto podemos crear diferentes programas en Python para poder y en este caso crear diferentes controladores.

6) Para la creación de los diferentes controladores para el movimiento de nuestro robot para ir de un punto A a un punto B, tomamos ayuda del siguiente video:

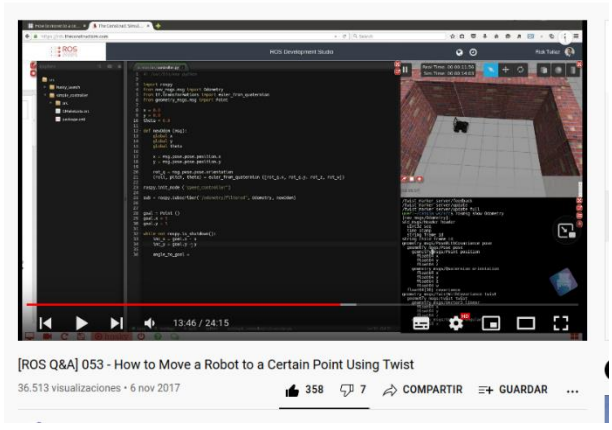


Fig.21
Video tomado de: https://www.youtube.com/watch?v=eJ4QPrYqMlw&list=LL&index=3&t=1215s

En este video se nos comenta de cómo se puede operar un robot móvil de 4 ruedas de un punto de coordenadas a otro punto de coordenadas por medio de un controlador hecho en Python por lo que es necesario crear un archivo Python.
También aprendimos a controlar la odometria de nuestro robot para saber en dónde se encuentra en el mapa, también a crear un suscriptor para el nombre de los topic-list que en este caso es el de (odometry). Cada ros-topic tiene un mensaje de salida que están compuestos por diferentes mensajes que se pueden apreciar en el caso de la odometria se importa poniendo (from nav_msgs.msg import Odometry) cabe resaltar que no todo los ros-topic manejan el mismo sistema de variables para el control de sus operaciones por lo que debemos estar pendientes a la estructura además de la manera de que están constituidas para poder crear un controlador con las dimensiones de las variables correctas para asi poder garantizar un buen funcionamiento del mismo al momento de ingresar valores por medio de la consola.

```python
import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import  Odometry
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import Point
from math import atan2,sqrt,pow,pi
import time

x=0
y=0
theta=0

msg=Odometry()

def position(msg):
    global x
    global y
    global theta
```

Fig.22

En nuestro caso para poder controlar la posición de nuestro robot y poderlo desplazar de un lugar A a un lugar B necesitamos controlar la posición (x, y, theta) de nuestro robot móvil.
Theta es algo más complejo de calcular en este caso debido que es la rotación de nuestro robot móvil en el eje z, pero la orientación de nuestro robot esta expresada por medio de un Cuaternion de orientación por lo que debemos utilizar la función de la línea 7que se encuentra arriba en nuestro código de Python al inicio para poder calcularla.
Mediante la información suministrada por medio de http://wiki.ros.org/tf2/Tutorials/Quaternions nos fue útil para poder comprender la función Quaternion en ROS para el control de la posición de nuestros robots debido a que ROS usa los cuaterniones para rastrear y aplicar rotaciones. Un Cuaternion tiene 4 componentes (x, y, z, w) por lo que la magnitud de un Cuaternion debería ser uno. Si los errores numéricos causan una magnitud de Cuaternion diferente a uno, ROS imprimirá advertencia.

También para poder controlar la articulación prismática de nuestro robot que es la parte que se encarga de alzar las estanterías para llevar los elementos de un lugar a otro.

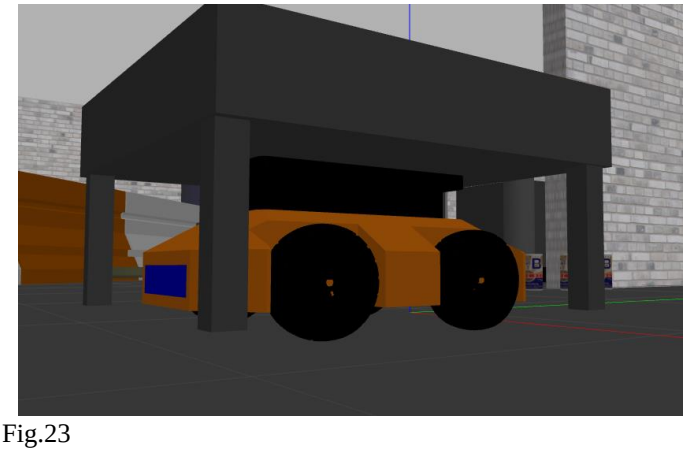
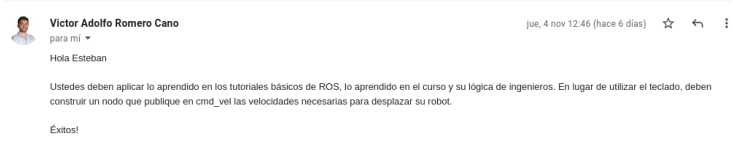


Fig.23
Imagen Simulación Articulación Prismática – Robot RAL.

Para poder controlar el movimiento de esta articulación seguimos la teoria explicada en la página de ROS.ORG la cual nos indica que podemos publicar diferentes mensajes por medio de ros pub pero también hemos consultado a nuestro profesor durante el desarrollo de este proyecto y él nos recomendó lo siguiente vía correo electrónico:



Víctor Adolfo Romero Cano — jue, 4 nov 12:46 (hace 6 días)
para mí

Hola Esteban

Ustedes deben aplicar lo aprendido en los tutoriales básicos de ROS, lo aprendido en el curso y su lógica de ingenieros. En lugar de utilizar el teclado, deben construir un nodo que publique en cmd_vel las velocidades necesarias para desplazar su robot.

Éxitos!

Fig.24

Por lo que decidimos investigar cómo hacer esto mismo, pero en este caso se diseñó un controlador para el movimiento prismático de la articulación por lo que hemos investigado mediante la siguiente página de origen chino: https://blog.csdn.net/baoli8425/article/details/117570072

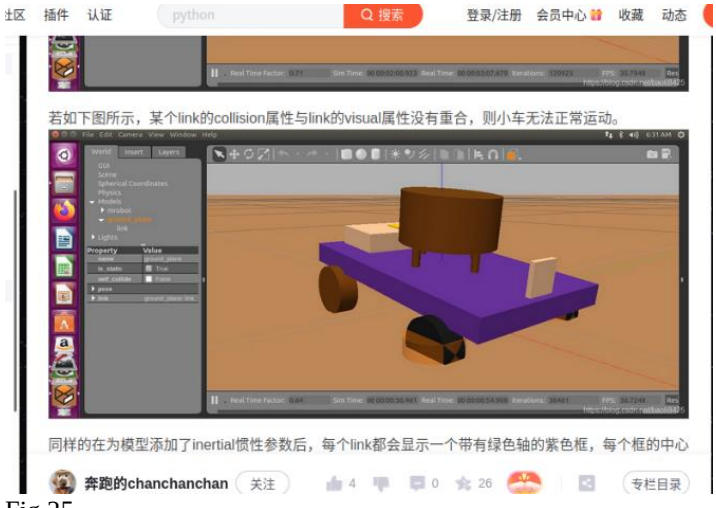


Fig.25

Por lo que hemos investigando aun mas sobre las articulaciones prismáticas nos encontramos con la información de que estas para que puedan alzar material se debe aplicar una fuerza en el archivo .URDF

```
    <parent
      link="base_link" />
    <child
      link="elevadorl" />
    <axis
      xyz="0 -1 0" />


    <limit lower="0" upper="0.5" effort="10000000" velocity="0.06"/>


  </joint>
  <link
    name="sensor2l">
    <inertial>
      <origin
        xyz="0.31183 -0.45 0.10864"
        rpy="0 0 0" />
      <mass
```

Fig.26

Cambien en otra pagina se nos indica de que se debe agregar un Plug-in a nuestra articulación prismática ademas de ponerle una etiqueta de transmisión para que este plug-in reconozca el link como una pieza de transmisión para así otorgar el movimiento

vertical para hacer posible el levantamiento de la estantería. Información encontrada en: https://www.theconstructsim.com/deal-transmission-tags-gazebo-ros-control/

```
<gazebo>
      <plugin name="gazebo_ros_control"
              filename="libgazebo_ros_control.so">
              <robotNamespace>/xe</robotNamespace>
      </plugin>
</gazebo>

<transmission name="${lr}_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="elevador">

         <hardwareInterface>EffortJointInterface</hardwareInterface>

      </joint>
      <actuator name="${lr}Motor">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>10</mechanicalReduction>
      </actuator>
    </transmission>
```

Fig.27

Además, se plantea que se debe poner un motor virtual el cual se encarga de simular el desplazamiento hacia arriba y hacia abajo de nuestro robot.

7) Para la creación de los mundos fue una etapa muy fácil debido a que había distintos videos en internet que nos ayudaban a crearlos mediante gazebo para después poner nuestro robot en estos mundos para asi poder simularlo en un ambiente virtual parecido a las condiciones de operación las cuales se estén solicitando.

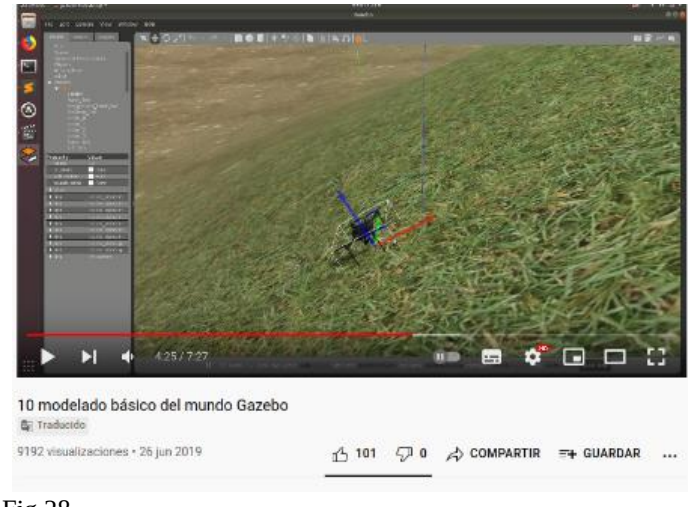


Fig.28
Video visto para la realización del mundo virtual robot RAL: https://www.youtube.com/watch?v=gpk8mQhhI4s&t=179s

Siguiendo las instrucciones de este video y también las recomendaciones de nuestro profesor durante las clases vistas durante el semestre hemos obtenido el siguiente mundo:

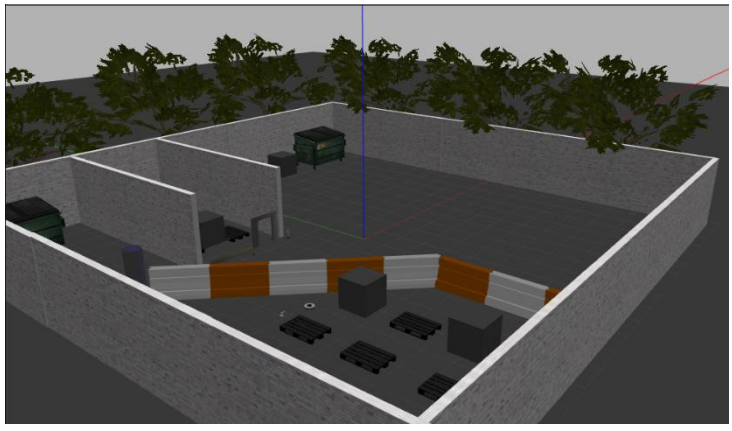


Fig.29
Mundo creado en Gazebo Robot RAL

Ya teniendo el robot creado hemos puesto a simular nuestro robot dentro del mundo virtual para observar como este se comporta.

Fig.30
Robot RAL Simulado dentro del mundo creado

8) Viendo la necesidad de poder insertar una estantería que tenga las dimensiones optimas para nuestro robot RAL, fue necesario investigar herramientas de diseño en ubuntu 20.04 ademas de contar con una compactibilidad con gazebo para poder ponerlas en la simulación y analizar los resultados por lo que con ayuda del siguiente video (https://www.youtube.com/watch?v=h4hZzPCOMKs) hemos obtenido los siguientes resultados:

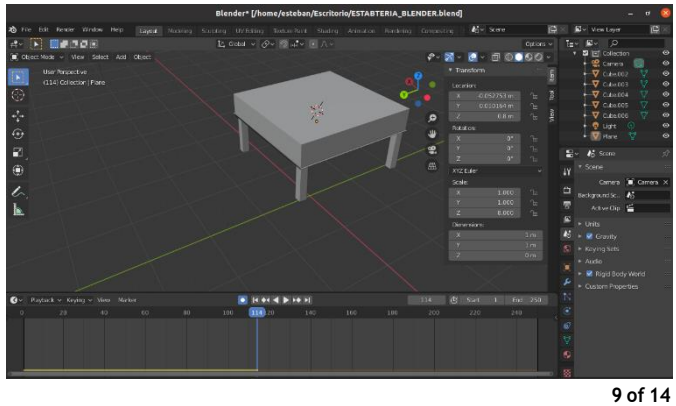


Fig.31
Diseño de estantería estándar para robot RAL -BLENDER

Investigando en diferentes páginas y videos hemos encontrado de que Blender es un excelente editor de modelado 3D para importar sólidos y piezas en gazebo además de que son simulables por lo que hemos realizado las estanterías en este programa y al mismo tiempo importarlas junto a gazebo para así mirar su simulación.

Fig.32
Imagen Simulación: Gazebo

En este apartado ya se puede apreciar a nuestro robot RAL en el entorno virtual que se ha creado anteriormente para ver su funcionamiento. Se han obtenido resultados positivos debido a que el robot se puede desplazar con facilidad y sin errores de movimiento al momento de desplazarse por el mapa y el suelo del mismo por lo que también procederemos a analizar nuestro robot en RVIZ.

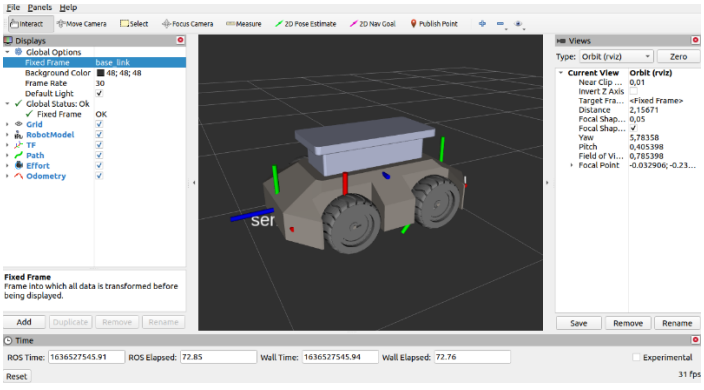


Fig.33
Simulación RVIZ robot RAL

Antes de seguir fue necesario investigar un poco sobre los archivos STL debido a que es el formato que Gazebo soporta para poder importar diferentes piezas y que estas sean simulables para su entorno. Para ello nos hemos basado en la siguiente página: https://es.3dsystems.com/quickparts/learning-center/what-is-stl-file

Como bien se sabe las piezas deben tener un eje de coordenadas y al mismo tiempo una superficie solida donde las cuales van a hacer colisión con el mundo virtual por lo que es necesario definir estos limites en nuestro archivo STL por medio de blender.

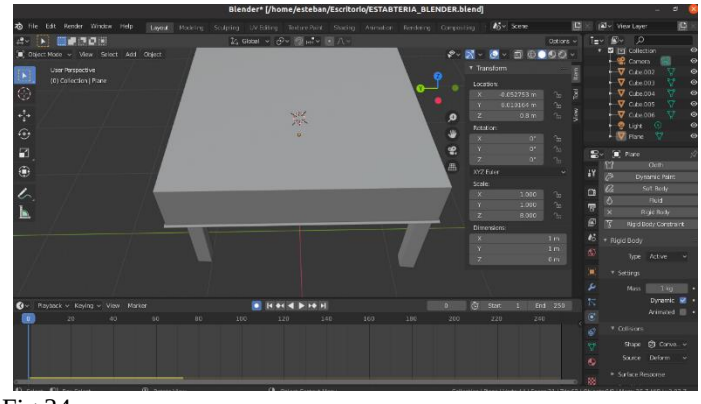


Fig.34

En este apartado se definen las inercias y las colisiones de nuestra estantería para que esta pueda ser bien simulada en nuestro entorno de gazebo además de permitir de que nuestro robot RAL pueda interactuar con el sistema de elevación de nuestro robot RAL.

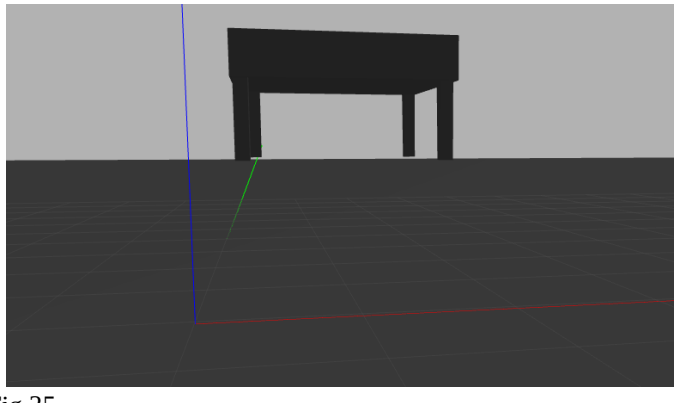


Fig.35

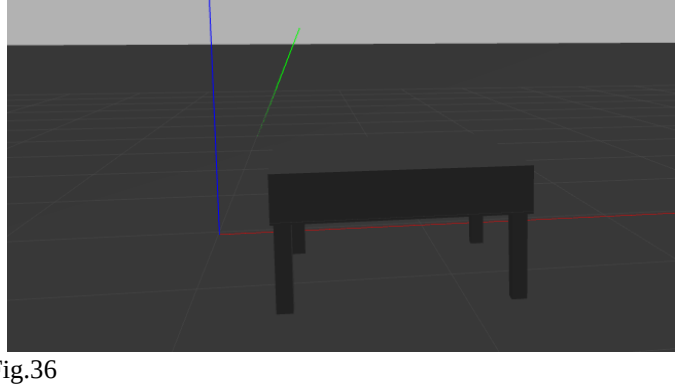


Fig.36

En este caso se realizó un ensayo de gravedad para verificar que nuestra pieza caiga en un entorno real y como se puede ver en las imágenes anteriores funciona correctamente por lo que se procede a hacer un ensayo en el entorno de gazebo para obtener los

resultados esperados que es levantar y dejar la estantería.

Fig.37

Fig.38

Finalmente podemos ver que nuestra pieza esta en conjunto con nuestro robot RAL y cumple con la función de alzar y dejar caer la pieza como parte del objetivo de nuestro proyecto.

En este apartado explicaremos los controladores, técnicas y algunos códigos que hemos utilizado para la realización eficaz de este proyecto.

Controlador de posición

Para poder hacer el posicionamiento de nuestro robot en el entorno virtual que hemos creado es necesario saber los mensajes que son publicados mediante el comando rostopic donde se puede apreciar los distintos mensajes que se envían.

```
esteban@estebitan:~/catkin_ws$ rostopic list
/clock
/cmd_vel
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/joint_states
/odom
/rosout
/rosout_agg
/tf
/tf_static
/xe/elevador_position_controller/command
/xe/elevador_position_controller/pid/parameter_descriptions
/xe/elevador_position_controller/pid/parameter_updates
/xe/elevador_position_controller/state
/xe/joint_states
```

Fig.39

Como se puede apreciar en la imagen anterior los rostopic list que se encuentran ejecutando en nuestro gazebo mientras estamos ejecutando la simulación. En este caso nos centraremos en /cmd_vel el cual nos permite controlar la velocidad y posición de nuestro robot en el mundo virtual.

Analizando este rostopic hemos determinado de que debíamos encontrar alguna forma de como poder cambiar los valores de alguna forma por lo que nos dimos en la tarea de encontrar algún tipo de controlador o un Plug-in para realizar esta tarea.

```
<gazebo>

<plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
  <updateRate>10</updateRate>
  <robotNamespace>/</robotNamespace>
  <leftFrontJoint>r1</leftFrontJoint>
  <rightFrontJoint>r2</rightFrontJoint>
  <leftRearJoint>r3</leftRearJoint>
  <rightRearJoint>r4</rightRearJoint>
  <wheelSeparation>0.4</wheelSeparation>
  <wheelDiameter>0.1</wheelDiameter>
  <robotBaseFrame>base_link</robotBaseFrame>
  <torque>20</torque>
  <topicName>cmd_vel</topicName>
  <broadcastTF>false</broadcastTF>
 </plugin>
</gazebo>
```

Fig.40

Aquí hemos creado un Plug-in para con la instrucción skid_steer_drive_controller → libgazebo_ros_skid_steer_drive.so.

Aquí se definen la taza de refresco que va a tener nuestro Plug-in entre mandar el comando que modifica el topic en este caso es el /cmd_vel, también se definen los Joints en nuestro que en nuestro caso son las ruedas denominadas r1, r2, r3. r4 al mismo tiempo de la separación de las llantas del base link. Con esto lograremos la tele-operación de nuestro robot por medio de teclado para que el usuario tenga una opción de control manual en caso de que el terreno en donde se debe mover el robot es desconocido o solamente para testear su funcionamiento antes de ponerlo en operación.

Para poder mover la articulación prismática que tiene nuestro robot para poder alzar las estanterías se indago de que este tipo de links en gazebo se pueden mover mediante el rostopic link_states el cual nos indica el estado de los links que se encuentran dentro de la simulación de gazebo. Indagando en internet se encontró de

que se necesita un Plug-in de articulaciones.

```
<gazebo>
      <plugin name="gazebo_ros_control"
              filename="libgazebo_ros_control.so">
              <robotNamespace>/xe</robotNamespace>
      </plugin>
</gazebo>

<transmission name="${lr}_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="elevador">

         <hardwareInterface>EffortJointInterface</hardwareInterface>

      </joint>
      <actuator name="${lr}Motor">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>10</mechanicalReduction>
      </actuator>
    </transmission>
```

Fig.41

Con este Plug-in es posible mover las articulaciones tipo prismáticas, pero adicionalmente hay que poner un actuador que es el encargado de realizar el movimiento vertical ademas de poderlo visualizar en gazebo.

```
xe:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  elevador_position_controller:
    type: effort_controllers/JointPositionController
    joint: elevador
    pid: {p: 100.0, i: 0.01, d: 1.0}
```

Fig.42

Para poder ejecutar correctamente nuestro Plug-in para poder mover la articulación prismática donde se define un Publish para los estados de los Joint de nuestro robot y se define a una frecuencia de 50 para publicar el estado de los Jount por lo que se va a realizar un controlador de posicion effor_controllers/JointPositionController y al tipo de joint que se desea controlar en nuestro caso es el elevador.

```
esteban@estebitan: ~
esteban@estebitan:~$ rostopic pub /xe/elevador_position_controller/comma
sgs/Float64 "data: 0.22"
```

Fig.43

Con este comando estamos modificando e invocando el controlador de posicion para nuestro elevador a un valor deseado por el usuario donde podemos subir o bajar y en gazebo tenemos los siguientes resultados.

Fig.44

Para poder mover nuestro robot mediante indicación de puntos por el usuario para que este siga una trayectoria propuesta se necesito un programa en pyhon el cual se encarga del movimiento de nuestro robot. Para esto fue necesario tener la cinemática de nuestro robot móvil para poder entender el movimiento y crear un programa que nos de la posibilidad de mover a nuestro robot desde un punto A a un punto B.

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

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import  Odometry
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import Point
from math import atan2,sqrt,pow,pi
import time
```

Fig.45

Para poder realizar este controlador de trayectorias es necesario importar algunas precedencias para poder realizar las operaciones requeridas como es el caso de rospy debido a que es una biblioteca de Pyhon del cliente de ros que proporciona la interfaz necesaria para la programación de Python.

La dependencia geometry_msgs.msg import Twist proporciona mensajes para primitivas geométricos comunes como puntos, vectores y poses. Estas primitivas están diseñadas para proporcionar un tipo de datos común y facilitar la interopereabilidad en todo el sistema.

La dependencia nav_msgs.msg import Odometry representa una estimación de la posición y la velocidad en el espacio libre.

```python
x=0
y=0
theta=0

msg=Odometry()

def position(msg):
    global x
    global y
    global theta

    x=msg.pose.pose.position.x
    y=msg.pose.pose.position.y
    ora_q=msg.pose.pose.orientation
    (roll,pitch,theta)=euler_from_quaternion([ora_q.x,ora_q.y,ora_q.z,ora_q.w])

rospy.init_node('Speed_Controller')
sub=rospy.Subscriber('/odom',Odometry,callback=position)
pub=rospy.Publisher('/cmd_vel',Twist,queue_size=1)
rospy.sleep(0.3)
r=rospy.Rate(4)

#to set a goal
goal=Point()

goal.x= 6

goal.y= 6
speed=Twist()

while not rospy.is_shutdown():
    x_diff=goal.x-x
    y_diff=goal.y-y
```

Fig.46

Como se puede apreciar en la imagen anterior se incorporan las variables x,y, y theta que son la que se pueden analizar en la cinemática de nuestro robot al mismo tiempo se controla la velocidad y posición de nuestro robot por medio de los mensajes que se van publicando y mandando por los rostopic. Al mismo tiempo saeteamos la posición deseada por el usuario y se definen en x,y.

```python
while not rospy.is_shutdown():
    x_diff=goal.x-x
    y_diff=goal.y-y

    distance=sqrt(pow(y_diff,2)+pow(x_diff,2))
    angle_to_go=atan2(y_diff,x_diff)
    if abs(angle_to_go-theta)>0.1:


        speed.linear.x =0
        speed.angular.z=1.8 if (angle_to_go-theta)>0 else -0.7
        #ROTACION

    elif distance>0.1:


        speed=Twist()
        speed.linear.x =5.0
        speed.angular.z=0.0

    elif distance <= 0.1:
```

Fig.47

Aquí definimos la velocidad lineal y la velocidad angular en la que va a ir nuestro robot ademas de la siguiente función:

```
while not rospy.is_shutdown():
    x_diff=goal.x-x
    y_diff=goal.y-y

    distance=sqrt(pow(y_diff,2)+pow(x_diff,2))
    angle_to_go=atan2(y_diff,x_diff)
    if abs(angle_to_go-theta)>0.1:


        speed.linear.x =0
        speed.angular.z=1.8 if (angle_to_go-theta)>0 else -0.7
        #ROTACION

    elif distance>0.1:
        speed=Twist()
        speed.linear.x =5.0
        speed.angular.z=0.0

    elif distance <= 0.1:

        t=Twist()
        pub.publish(t)
          t2 = time.time()


        goal.x = 0

        goal.y = 0

    pub.publish(speed)
    r.sleep()
```

calcula la diferencia entre la meta propuesta por el usuario vz la posición del robot en el mapa ademas del angulo entre el punto deseado en base a la posición del robot en del robot. Si esta es igual el robot modificada la velocidad angular hasta que el angulo sea mayor a 0.1 entre el punto propuesto por el usuario y seguirá su trayectoria hasta llegar a su punto de llegada.

DISCUSIÓN

El proyecto tuvo como objetivo diseñar un robot AGV de logística para poder ser utilizado en ambientes de trabajo de bodegaje. se pudo lograr el objetivo gracias a la utilización de los conocimientos adquiridos durante la materia y anteriores se pudo realizar el modelo en solidworks, gracias a la utilización de ros, Rviz y gazebo se pudo realizar el modelamiento del robot y simulación del robot. Con ayuda de los modelos cinemáticos directos e inversos. Se logró hacer que el robot siguiera una ruta establecida teniendo en cuenta unas condiciones dadas se pudo identificar que el robot 4wd se puede asemejar al comportamiento de un robot diferencial. Tomando como inspiración el robot kiva se logró hacer un buen modelo que podría llegar a competir con este en un futuro, gracias a que su fabricación sería más económica pudiendo hacer que esta tecnología impacte mucho más en el sector de logística, haciendo que muchas empresas aumenten considerablemente su productividad al obtener los productos de una forma mas rapida y eficiente, mejorando el tiempo de producción y entrega de estos mismos.

CONCLUSIONES

- Durante el desarrollo de este proyecto se logro un aprendizaje significativo de todo lo aprendido durante las clases, y enfocado a un problema de la realidad esto da un enfoque de ingenria de como lo aprendido tiene un impacto en una problemática, logrando una solución efectiva para la problemática planteada.
- Podemos concluir que todas las herramientas vistas durante el curso de robotica son una fuerte base para este tipo de acercamientos hacia la robotica, para posteriores

cursos o proyectos a desarrollar, además de que crea un pensamiento para el desarrollo de robots móviles con esto poder generar soluciones en lo problemas que se presente y tener las capacidades desde la ingeniería de resolverlos.

- Nosotros como estudiantes aprendimos a manejar correctamente las fuented de información que se encuentra en internet debido a que durante la realización de este proyecto necesitamos investigar muchas cosas en internet y fue muy necesario aprender a utilizar las diferentes herramientas de búsqueda que se encuentran.
- Los robots AGV son muy útiles para el ambiente de la insdustrializacion y la manufactura debido a que la reducción de accidentes por lesiones físicas en este tipo de lugares se vera altamente reducida, además las empresas optimizaran sus ganancias de producción además de aumentar sus tasas de envíos por hora al optar por una tecnología de automatización como la que incorpora el robot RAL que hemos creado para este proyecto.
- Los links y Joints son conceptos que se deben tener muy claros antes de comenzar a hablar sobres robots debido a que son partes elementales además de que pueden ser controladas dependiendo de las restricciones de movimiento que tengan respectivamente.
- Los robots Moviles son limitados mediante el tipo de llantas que posean adeams de su configuración base por lo que cambiando la configuración de las llantas de los robots móviles podremos alterar su movilidad o hasta mejorarla.
- Los sensores son esenciales para el manejo y control de robots debido a que son los encargados de ayudanos a persevir lo que esta viendo nuestro robot para así evitar acidenter o tener un control de movimiento de nuestros robots.
- Para la búsqueda de información sobre este proyecto que hemos aprendido sobre la importancia del ingles debido a que la mayoría de la información de robótica se encuentra en ingles además los mejores libros de robótica están escritos en ingles.

REFERENCIAS

[1]Umit Blinge, Bogazici. (1997). AGV system with multi-load carriers: basic issues and
potential benefits. UKACC International Conference on Control 2012Elsevier, Volumen 16/
no.3, 1.

[2]UNIVERSIDAD TECNOLÓCIGA DE PANAMÁ, «MODELADO Y CONTROL DEL ROBOT MÓVIL ROBOTNIK SUMMIT XL,» Panamá, 2016.

[3]es - ROS Wiki. (s. f.). Ros.org. Recuperado 12 de octubre de 2021, de http://wiki.ros.org/es

[4]10 Basic Gazebo World Modeling. (2019, 27 junio). [Vídeo]. YouTube:https://www.youtube.com/watch?v=gpk8mQhhI4s&t=179s

[5]ROS | Gazebo仿真—阿克曼（Ackermann）四轮小车模型 chanchanchan的博客-CSDN博客阿克曼小车. (s. f.). CSDN. Recuperado 1 de noviembre de 2021, de [6]https://blog.csdn.net/baoli8425/article/details/117570072

Gazebo : Tutorial : Make a Mobile Robot. (s. f.). gazebosim. Recuperado 25 de octubre de 2021, de [7]http://gazebosim.org/tutorials/?tut=build_robot

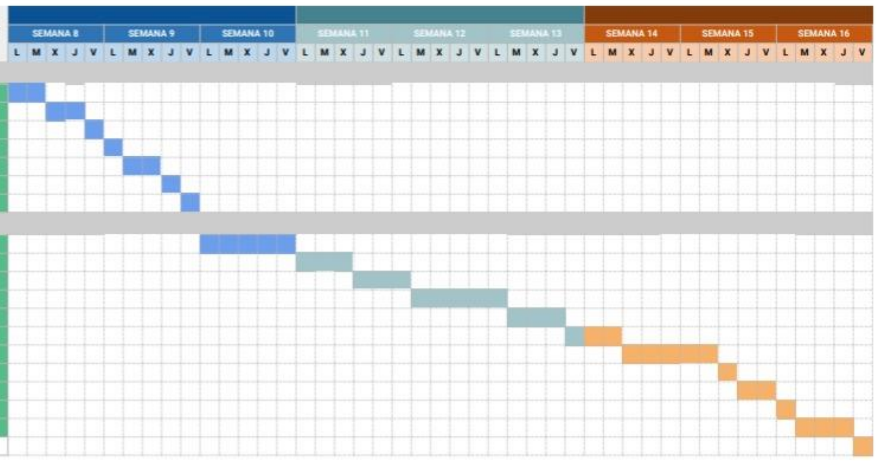
LA GUÍA DEFINITIVA DE BLENDER! (Tutorial completo en Español) | Desde cero! 2.91. (2020, 24 agosto). [Vídeo]. YouTube. https://www.youtube.com/watch?v=h4hZzPCOMKs

[8] ¿Qué es un archivo .STL? (s. f.). 3D Systems. Recuperado 2 de noviembre de 2021, de https://es.3dsystems.com/quickparts/learning-center/what-is-stl-file

ANEXO

Robot automatico de logistica

NÚMERO EDT	TÍTULO DE LA TAREA	FECHA DE INICIO	FECHA DE ENTREGA	DURACIÓN	% COMPLETADO DE LA TAREA
1	**Primera entrega**				
1.1	Planteamineto de problema	13/09/21	15/09/21	2	100 %
1.1.1	Planteamiento solucion	16/09/21	18/09/21	2	100 %
1.2	Modelado solidworks y visualizacion Rviz	18/09/21	19/09/21	1	100 %
1.3	Simulacion	20/09/21	21/09/21	1	100 %
1.4	Teleoperacion	21/09/21	23/09/21	2	100 %
1.5	Presentacion	23/09/21	24/09/21	1	100 %
1.6	Entrega	24/09/21	25/09/21	1	100 %
2	**Entrega Final**				
2.1	Mejora del modelo de solidworks	27/09/21	01/10/21	5	100 %
2.2	Asignacion de material	01/10/21	04/10/21	3	100 %
2.3	Mejora URDF	04/10/21	07/10/21	3	100 %
2.4	Redimensionamiento	08/10/21	14/10/21	5	100 %
2.5	Fase de investigacion	15/10/21	21/10/21	4	100 %
2.6	Planeacion de controlador	21/10/21	25/10/21	3	100 %
2.7	Programacion controlador	26/10/21	01/11/21	5	100 %
2.8	Indagacion moviento prismatico	02/11/21	03/11/21	1	100 %
2.9	Programcion controlador moviento prismatico	03/11/21	05/11/21	2	100 %
2.10	Creacion mundo y programacion entorno	08/11/21	09/11/21	1	100 %
2.11	Simulacion	09/11/21	11/11/21	2	100 %
2.12	Entrega	11/11/21	12/11/21	1	0 %



Anexo#1