Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Puebla

Fundamentación Robótica TE3001B Gpo(101)

Diseño de un sistema en ROS para controlar un motor de corriente continua

Alumnos

Diego García Rueda
A01735217

Jhonatan Yael Martínez Vargas
A01734193

Jonathan Josafat Vázquez Suárez
A01734225

Profesores

Dr. Alfredo García Suárez

Dr. Rigoberto Cerino Jiménez

Dr. Juan Manuel Ahuactzin Larios

18 abril 2023, Puebla, México.

Resumen

En este informe se abordará el uso de Gazebo, un simulador de código abierto que permite la creación de modelos de robots y la simulación de su comportamiento en entornos virtuales. Gazebo es una herramienta fundamental para el testeo de algoritmos de control, el desarrollo de aplicaciones y el entrenamiento de robots en diferentes escenarios, todo esto de manera segura y económica. Además, gracias a su interfaz gráfica de usuario, Gazebo facilita el proceso de diseño y desarrollo de sistemas de robótica al permitir al usuario realizar pruebas y ajustes en tiempo real.

Este informe también se enfocará en la comunicación SSH para el control del Puzzlebot, un robot modular y personalizable que permite a los usuarios crear configuraciones de robot adaptadas a sus necesidades. El uso de SSH permite configurar, monitorear y programar funciones en el Puzzlebot, lo que resulta en una programación más eficiente y una mayor flexibilidad en la creación de configuraciones personalizadas de robots.

En resumen, este informe tiene como objetivo explorar el uso de Gazebo y SSH para el control del Puzzlebot, con el fin de demostrar cómo estas herramientas pueden ser utilizadas de manera efectiva en el desarrollo de sistemas de robótica.

Objetivos

Comprender el funcionamiento del Puzzlebot, tanto a nivel de hardware como de software mediante ROS, aprender a utilizar protocolos de intercomunicación como SSH y Master URI de ROS, monitorear y controlar en tiempo real los movimientos del robot así como una simulación en Gazebo.

Introducción

Gazebo es un simulador de código abierto que permite la creación de modelos de robots y la simulación de su comportamiento en entornos virtuales. Es una herramienta fundamental para el testeo de algoritmos de control, el desarrollo de aplicaciones y el entrenamiento de robots en diferentes escenarios, todo esto de manera segura y económica. Además, gracias a su interfaz gráfica de usuario, Gazebo facilita el proceso de diseño y desarrollo de sistemas de robótica al permitir al usuario realizar pruebas y ajustes en tiempo real.

ROS Master URI es un identificador único que se utiliza en el framework ROS (Robot Operating System) para la comunicación entre diferentes componentes de un sistema de robótica. Es una especie de dirección web que permite que los nodos ROS se encuentren y se comuniquen entre sí a través de un canal seguro. Además, el ROS Master URI también se utiliza para la gestión de recursos compartidos entre diferentes nodos, como sensores, actuadores y datos.

Una vez declarado el funcionamiento y comportamiento de Gazebo junto a ROS Master URI, podemos remotamente añadir un control mediante el protocolo SSH, una herramienta que permite acceder y controlar sistemas de manera remota a través de una conexión encriptada. El protocolo SSH permite al usuario configurar, monitorear y programar funciones. Esta

herramienta es principalmente funcional en el control del puzzlebot con funciones como roslaunch, roscore, rosrun y rqt.

Puzzlebot es un robot modular y personalizable que permite a los usuarios crear configuraciones de robot adaptadas a sus necesidades. El robot consta de módulos básicos, como ruedas, sensores y actuadores, que se pueden ensamblar de diferentes maneras para crear diferentes configuraciones de robot.

Solución del problema

Para la solución de este problema se decidió utilizar los paquetes proporcionados por la organización socio formadora, estos paquetes fueron: "puzzlebot_control", "puzzlebot_gazebo" y "puzzlebot_world", con estos archivos nos permiten realizar simulaciones en Gazebo en las que ya se encuentra un modelo 3D del robot usando el simulador.
El equipo desarrolló un nodo en Python que a partir de un path ingresado por el usuario punto por punto el programa controla al robot permitiendo seguir esa ruta, este recorrido se ve en la simulación al igual que de manera física mediante una comunicación SSH, este programa controla al robot mediante un sistema de lazo abierto, el código será explicado a profundidad en la siguiente sección.

Metodología realizada

En el desarrollo del nodo decidimos crear una implementación en la que dependiendo de la posición en la que se encuentre el robot, las velocidades lineales y angulares del robot cambiarán. El nodo se suscribe a los nodos "wl" y "wr", el cambio de las velocidades se publica mediante un *Twist(),* este es un mensaje personalizado de ROS, que contiene los desplazamientos y rotaciones en los 3 ejes.

El programa solicita al usuario ingresar el número de puntos a alcanzar junto a sus coordenadas y la velocidad de desplazamiento. Haciendo uso del siguiente código creamos una posición actual y objetivo, donde calcula el ángulo y distancia a recorrer.

```
x1, y1 = self.route[actual_id]
x2, y2 = self.route[target_id]

xx, yy = (x2 - x1, y2 - y1)
distance = math.sqrt(xx**2 + yy**2)
buffer = math.atan2(xx, yy)
rotation = buffer - last_rot
last_rot = buffer
```

Primero, al solicitar un punto al usuario, extraemos las coordenadas (x,y) y las guardamos como nuestro objetivo. Luego, para calcular la distancia entre los dos puntos, utilizamos el teorema de Pitágoras para obtener la magnitud. Para el ángulo de rotación, usamos la función trigonométrica del arcotangente y restamos la rotación actual del robot. Es importante destacar que, una vez que llegamos al punto deseado y calculamos nuestros movimientos futuros, reiniciamos nuestras variables de traslación y rotación.

El proceso de desplazamiento del robot se realiza en tres etapas. En primer lugar, el robot se detiene y evalúa la siguiente acción que debe tomar. A continuación, el robot gira hasta alcanzar el ángulo de giro deseado, utilizando la función trigonométrica del arcotangente para determinar el ángulo de rotación necesario. Finalmente, el robot avanza linealmente hacia el punto deseado. Este enfoque de desplazamiento por etapas permite que el robot se mueva de manera más precisa y eficiente a través de la ruta. Además, el reinicio de las variables de traslación y rotación una vez que se alcanza el punto deseado asegura que el robot esté listo para realizar el siguiente movimiento.

Para asegurarnos que el robot logre encontrar la rotación deseada, añadimos un umbral al rango, este es de ± 0.6 grados, lo que equivale a 0.0104. *self_orientation* se modificará de acuerdo a la siguiente fórmula utilizando parámetros como el radio de las ruedas del robot, distancia entre ambas ruedas y la velocidad angular de cada rueda y un diferencial de tiempo.

Durante esta etapa de rotación únicamente existirá velocidad angular en el eje Z (en este eje se realiza la rotación) mientras que se detendrá su velocidad lineal. Estas velocidades se publican al robot.

```
   current_time = rospy.get_time()
   dt = current_time - last_time
   last_time = current_time

   if self.orientation > rotation + 0.01 or self.orientation < rotation -
0.01:

      self.orientation += float(self.wheel_radio * ((self.angular_rigth -
self.angular_left) / self.wheel_distance) * dt)

   msg.angular.z = 0.1
   msg.linear.x = 0.0
   self.velocity_pub.publish(msg)
```

Posteriormente de haber alcanzado el ángulo deseado se procede al avance lineal, este se calcula conociendo la distancia que ya ha avanzado el robot, para esto utilizamos una fórmula que utiliza el radio de las ruedas y su velocidad lineal. En caso de que la distancia recorrida sea menor a la distancia deseada el robot seguirá avanzando hasta llegar al punto. A diferencia de la etapa anterior, aquí se detiene la velocidad angular y la velocidad lineal en el eje X (el frente del robot) adquiere el valor de velocidad ingresado por el usuario. Estas velocidades son publicadas al robot.

```
   elif self.distance_traveled < distance:

   self.distance_traveled += float(self.wheel_radio * ((self.angular_rigth +
                                          self.angular_left) / 2) * dt)
```

```
msg.angular.z = 0.0
msg.linear.x = self.linear_velocity

self.velocity_pub.publish(msg)
```

Después de que el robot alcanza el punto deseado, aumentamos el valor de *target_id* en uno, lo que indica el siguiente punto en la trayectoria. Además, reiniciamos nuestras variables de distancia recorrida y rotación del robot para prepararnos para el siguiente movimiento. Si el punto alcanzado es el último de la ruta, el robot se detendrá y el programa finalizará su ejecución.

```
else:
    target_id += 1
    self.distance_traveled = 0.0
    self.orientation = 0.0



    if target_id >= route_size:
           self.stop()
            rospy.signal_shutdown("Route Completed")
            self.rate.sleep()
```

En la ejecución del programa

Resultados

Link al video de la simulación en Gazebo:

Nota: Se recomienda ver el video en velocidad x1.5

https://youtu.be/foMiGt5GyAA

Link al video del puzzlebot real:

Nota: Se recomienda ver el video en velocidad x1.5

https://youtu.be/foMiGt5GyAA

Conclusiones

La implementación de protocolos de intercomunicación, como SSH y ROS Master URI, fue fundamental para el control remoto del robot y el monitoreo en tiempo real de sus movimientos y acciones. Además, el desarrollo de un algoritmo base de pathfinder para controlar un Puzzle Bot permitió no solo lograr un control eficiente de la navegación del robot en un ambiente

complejo sino que abre puertas a un desarrollo más avanzado y oportunidad de trabajar en muchas mejoras de inteligencia propia. En resumen, esta primera práctica generó grandes conocimientos en el control de sistemas operativos, el manejo de ROS con Master URI y la simulación de gazebo.