Taller 4 Robótica

Santiago Martínez Castaño
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá
s.martinezc@uniandes.edu.co

Martin David Galvan Castro
Departamento de Ingeniería
de Sistemas y Computación
Universidad de los Andes, Bogotá
md.galvan@uniandes.edu.co

Santiago Montaño Díaz
Departamento de Ingeniería
Eléctrica y Electrónica
Universidad de los Andes, Bogotá
vs.montano@uniandes.edu.co

I. Introducción

El taller 4 tiene como objetivo la familiarización con un robot omnidireccional, en este caso el *Senecabot*, que es un robot omnidireccional de cuatro ruedas. Por otra parte, esta es la primera practica con la cual se cuenta con un robot físico, por lo que en esta práctica se verá el funcionamiento en condiciones no ideales, diferente a las simulaciones.

II. Desarrollo

II-A. ***Punto 1 - Control***

Para el desarrollo del control de posición fue necesario, primero obtener la posición actual del robot, y a partir de esta, calcular la diferencia entre la pose actual y la pose deseada, finalmente se desplaza en la trayectoria tal que el error se minimice.

En la implementación se creo el nodo *Senecabot_Position_Controller*, este nodo implementa un Publisher, con el cual se envía el perfil de velocidad al paquete *Senecabot_robot* a través del tópico */cmd_vel*, y además tiene un Listener, con el cual se obtiene la posición actual del robot a travez de lo publicado por el paquete *Senecabot_robot* en el tópico de */odom*.

Al obtener la posición del tópico */odom*, se tenia que hacer una conversión en la orientación del robot, ya que en el mensaje entrante la orientación estaba dada en cuaterniones, y el control se implemento para rotaciones ordenadas, para la conversión, se uso la función de *euler_from_quaternion()* del paquete tf de ROS.

Para la implementación del control del robot, se definió que se usaría un control proporcional (Las constantes fueron encontradas iterativamente, teniendo como objetivo las constantes que permitieran el movimiento más rápido y controlado del robot) y se creo una variable auxiliar *c*, que su funcionamiento ayuda a determinar al algoritmo si enviar un mensaje para generar un movimiento en el eje X o Y del robot. Primero, se parte de calcular el error en el eje X y Y y orientación global. A partir de esto se decide en que movimiento se va a hacer el control, acá es donde se usa la variable C, ya que si C no es divisible por 2 ($c \mod 2! = 0$) se va a hacer un desplazamiento en el eje X, si es divisible se hace un desplazamiento en el eje Y. Para actualizar la variable C, sin importar el movimiento se suma 1 a la variable C. Este proceso se repite hasta llegar a las coordenadas finales, donde se hace un control de la orientación para la orientación final deseada. En la figura 1 se puede ver el algoritmo implementado, y en la 2 se puede ver el grado de ROS de la comunicación entre el nodo y el paquete de *Senecabot_robot*

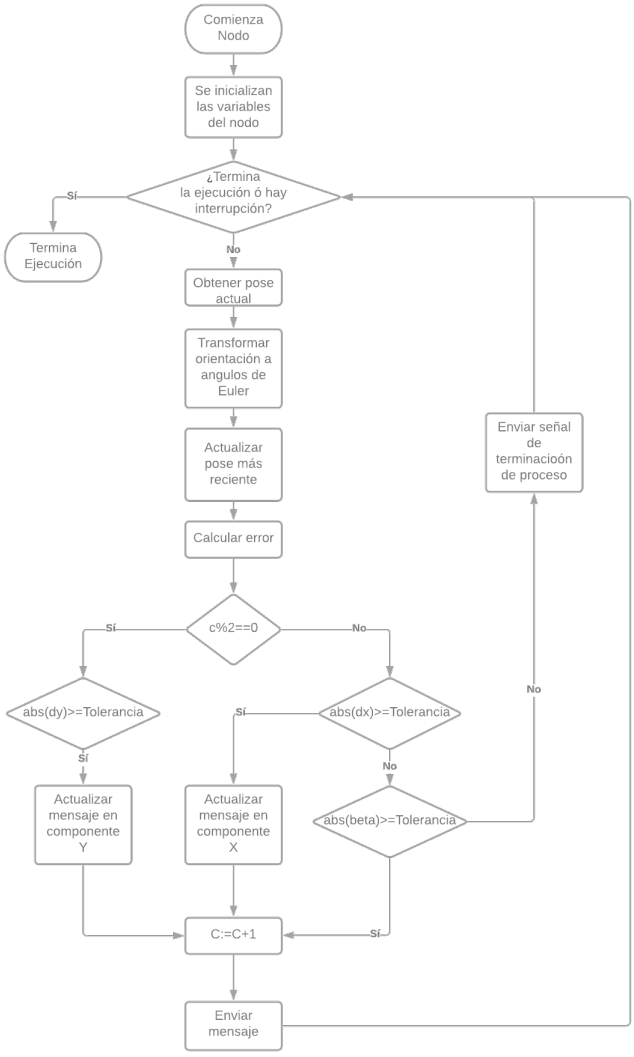

Figura 1. Algoritmo Senecabot_Position_Controller

II-B. ***Punto 2 - Figuras***

Para este punto, se desarrolló el nodo *Senecabot_Drawer*, este implementa el control desarrollado en el punto anterior. Gracias a que el control necesitaba como parámetros las constantes de ganancia y el punto final, entonces, se decidió hacer la siguiente aproximación:

- Discretizar las figuras, identificando las esquinas
- Mapear las esquinas dentro de un pliego de cartulina
- Guardar las coordenadas de las esquinas mapeadas en un archivo txt para su lectura

El funcionamiento de este nodo se basa en que se lee el archivo txt donde están almacenadas las coordenadas, estas se guardan en un arreglo. Las condiciones para acabar el control también cambian, por que se espera una interrupción o que el robot ya haya recorrido todas las coordenadas del archivo txt

Como resultado se obtuvo que el robot no se mueve de la manera esperada, ya que presenta deslizamiento durante su trayecto, lo cual dificulta el robot, entonces se decidió que para comprobar el correcto funcionamiento del algoritmo, se simulo un robot omnidirecional en V-REP, donde este mostró un comportamiento satisfactorio ya que las figuras al menos se distinguen, por lo que se podría decir que el control funciona, pero gracias a que durante la ejecución no se tienen las condiciones idealizadas de la simulación, no se obtiene el resultado esperado. En la figura 2 se muestra el algoritmo que se siguio:

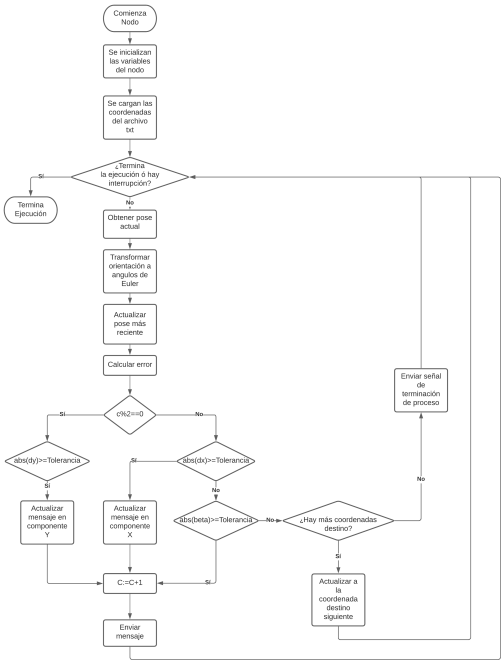

Figura 2. Algoritmo Seneabot_Drawer

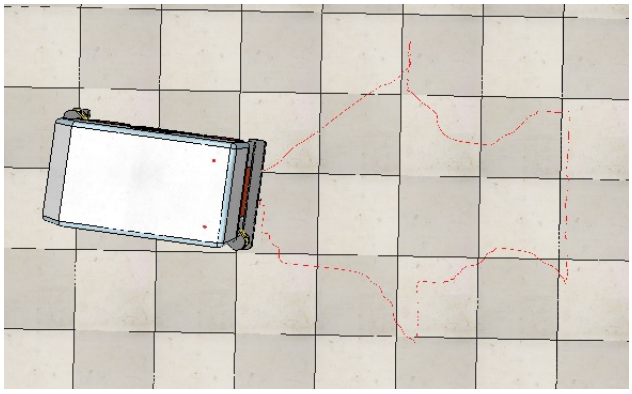

Figura 3. Figura Pez

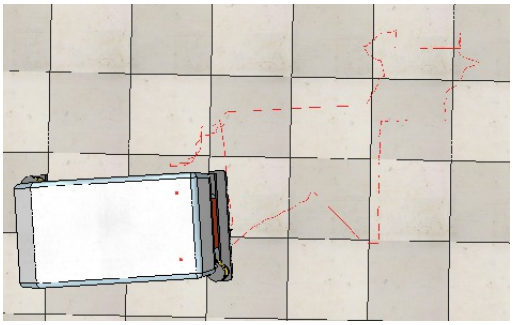

Figura 4. Figura Toro

II-C. ***Punto 3- Navegación***

Para este punto, primero se obtuvo un mapa del entorno del robot el cual se observa a continuación:

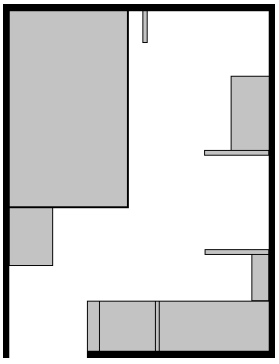

Figura 5. Mapa del entorno del robot

Se implemento el nodo *Senecabot_navigator* , el algoritmo implementado permite al usuario seleccionar el punto destino del robot en el mapa mediante clics. Una vez dada la pose final del robot, se procede a calcular la ruta óptima mediante el algoritmo de búsqueda A*, lo primero que se hace es expandir al nodo inicial en sus vecinos y se añade el nodo vecino con menor probabilidad, luego de esto se verifica si este nodo es el destino, si es así se añade a la lista de nodos visitados y se finaliza el algoritmo de búsqueda. Si el nodo no es el destino se procede a añadirlo a la lista de nodos visitados, se coloca

como nodo actual y se realiza el mismo procedimiento inicial. Este proceso se realiza iterativamente hasta que se llegue al nodo destino. Finalmente, cuando se tiene la ruta óptima con el algoritmo de búsqueda, se genera la misma en el mapa, y se procede a mover el robot hacia el punto destino. Para seguir los puntos de la ruta se implemento punto a punto el control del primer literal, la lógica del nodo se observa en el siguiente macro-algoritmo:

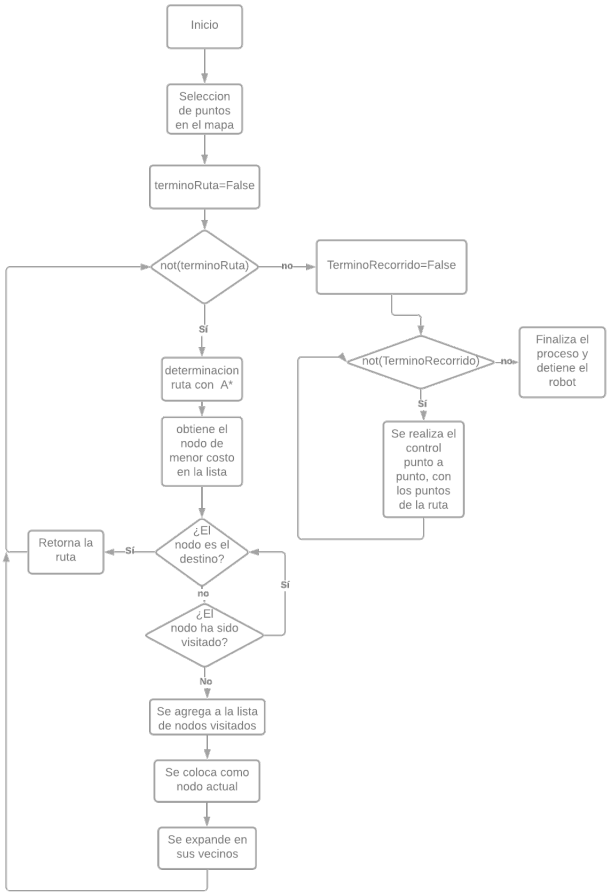

Figura 6. Macro-algoritmo *Senecabot_navigator*

A continuación se muestra la ruta obtenida en la implementación:

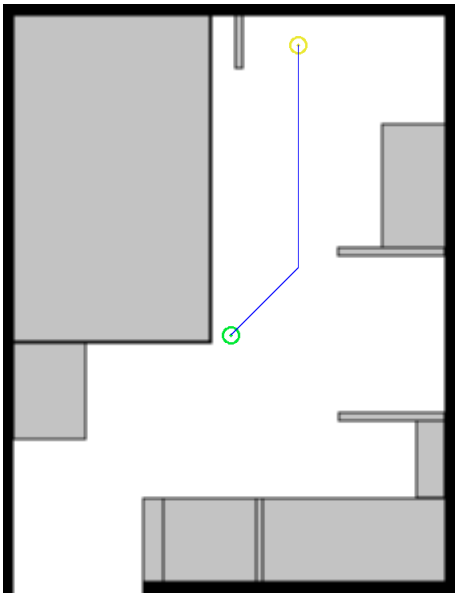

Figura 7. Ruta obtenida

III. Instrucciones de Ejecución

Para comprobar el funcionamiento de los nodos creados es necesario que el dispositivo que ejecuta la simulación disponga de herramientas de terceros que han sido empleadas en el código de esta solución. Posterior a esto, en la sección III-A, se explicará cómo ejecutar la simulación y comprobar el funcionamiento de los nodos desarrollados. Cabe aclarar que se debe usar python2 para ejecutar cada uno de los nodos.

III-A. Prueba de funcionamiento de los nodos

- Para la ejecución del nodo de control, primero es necesario hacer la configuración del Senecabot, después de esto, se debe iniciar el paquete *SenecaBot*, y finalmente se ejecuta el nodo, los comandos son:

```
$ roslaunch seneca_robot seneca_robot.launch
$ rosrun taller4_4 senecabot_position_controller.py [X,Y,Z]
```

  Donde X,Y,Z corresponden a los parametros de ubicación final del robot.
- Para la ejecución del nodo *Senecabot_Drawer*, primero es necesario hacer la configuración del Senecabot, después de esto, se debe iniciar el paquete *SenecaBot*, y finalmente se ejecuta el nodo, los comandos son:

```
$ roslaunch seneca_robot seneca_robot.launch
$ rosrun taller4_4 senecabot_drawer.py
```

- Para la ejecución del nodo *Senecabot_navigator*, primero es necesario hacer la configuración del Senecabot, después de esto, se debe iniciar el paquete *SenecaBot*, y finalmente se ejecuta el nodo, los comandos son:

```
$ roslaunch seneca_robot seneca_robot.launch
$ rosrun taller4_4 senecabot_navigator.py
```