PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC-2685 ROBÓTICA MÓVIL

LAB 2: CONTROL EN LAZO CERRADO Y COMPORTAMIENTOS REACTIVOS

Fecha de Presentación: Martes 17 de mayo de 2022

Introducción

Vamos avanzando en el control de los movimientos de los TurtleBots. En el Lab 1 aprendieron a mover el robot y a definir trayectorias (lista de poses) como una sucesión de distancias que fueron calculadas a partir de una velocidad predefinida y tiempos de aplicación (control en lazo abierto). Además, lograron programar una rutina de percepción básica para la de detección de obstáculos.

En esta oportunidad, daremos el siguiente paso en el control de los movimientos del robot, y lo dotaremos con la capacidad de cuantificar su desplazamiento a través de la odometría. En esta experiencia supondremos que la odometría es "perfecta", es decir, que no existe error entre la posición real del robot y la información entregada por el sensor. Para este laboratorio, programarán un conjunto de rutinas para las cuales el control realimentado es fundamental. Dichas rutinas consisten en: llevar al robot a una posición deseada minimizando el error, navegar por un pasillo complejo manteniendo la distancia a las paredes y seguir una trayectoria previamente definida.

1. Llegando a la posición deseada utilizando control P y PI

Como vimos en el laboratorio anterior, sin realimentación de la odometría los movimientos pueden ser inciertos al usar control en lazo abierto (sobre todo los giros). Esta vez, deberán programar movimientos de mayor precisión usando controladores en lazo cerrado, como el que se muestra en la figura 1.

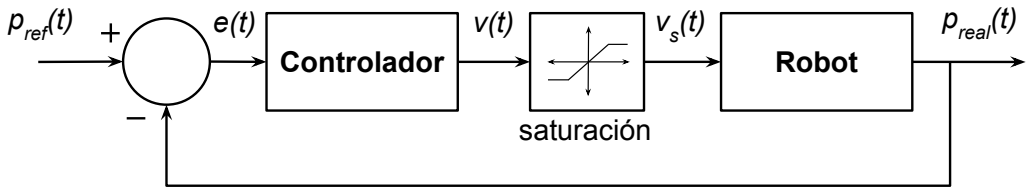

Figura 1: Diagrama de control en lazo cerrado con saturación en actuación.

Para diseñar el controlador, nos basaremos en modelos del tipo tipo P y PI y buscaremos los parámetros que mejor se ajusten a cada problema. Para ello, cada controlador será instanciado como un nodo de ROS, cuya implementación será provista por la biblioteca *PID* [1]. Para el caso de las señales a controlar ($P_{real}(t)$ en la figura 1), definiremos las variables *distancia* y *ángulo objetivo*, las cuales deberán seguir nuestra señal de referencia ($P_{ref}(t)$ en la figura 1).

A continuación se detallan las tareas específicas que debe programar:

- Programar una función **`giro_controlado(`**θ**`)`**, que reciba como argumento el ángulo al que debe girar el robot ($P_{ref}(t)$), y realizar el giro utilizando un controlador en lazo cerrado. Establezca una velocidad máxima de rotación de 0.7 [rad/s] a través de una función de saturación como la esquematizada

en la figura 1. Para ello, utilice los parámetros `lower_limit` y `upper_limit` del nodo *controller* [1].

- Programar una función **`desplazamiento_controlado(dist)`**, que reciba como argumento la distancia a la que el robot debe avanzar en línea recta, y realizar el desplazamiento utilizando un controlador en lazo cerrado. Establezca una velocidad máxima de desplazamiento de 0.3 [m/s] a través de una función de saturación como la esquematizada en la figura 1. Para ello, utilice los parámetros `lower_limit` y `upper_limit` del nodo *controller* [1].

- Usando las funciones anteriores, cree una función **`mover_robot_a_destino_ctrl(goal_pose)`** similar a la función que programó en el Lab 1, pero esta vez usando una secuencia de movimientos de giro y desplazamientos controlados para lograr las poses deseadas. El argumento `goal_pose` es una lista de poses en formato (x, y, θ). Todas las poses de la lista serán relativas a la posición inicial del robot, es decir, la lista de poses $[(1, 0, \pi/2), (1, 1, \pi), (0, 1, -\pi/2), (0, 0, 0)]$ hará que el robot describa un cuadrado de lado 1 [m].

Una vez programadas estas funciones, deberá configurar un controlador para la variable *distancia lineal objetivo*, y otro para la variable *ángulo objetivo*. Debido a que en la presente experiencia evaluaremos el funcionamiento tanto de controladores del tipo P como del tipo PI, en total deberá ajustar los parámetros de 4 controladores distintos como se resume a continuación:

- Control P: 1 controlador para distancia y 1 controlador para ángulo

- Control PI: 1 controlador para distancia y 1 controlador para ángulo

Para determinar el valor de los parámetros K_p y K_i, deberá seguir el método de *prueba y error* hasta alcanzar el menor error posible en la trayectoria descrita por el robot. Se recomienda comenzar a probar con valores bajos para ambos parámetros (del orden de 0.01), y posteriormente aumentarlos gradualmente hasta alcanzar un movimiento satisfactorio. Tenga la precaución de verificar que la actuación esta siendo correctamente limitada por la función de saturación.

Una vez finalizadas las etapa de implementación y de establecimiento de parámetros, deberá repetir la rutina *Avanzar y Rotar* solicitada en el Laboratorio 1. Estas deberán ser evaluadas tanto para el controlador P como para el controlador PI.

Incluir en presentación:

- Gráfico con señales de: referencia $P_{ref}(t)$, actuación $V_s(t)$ y salida $P_{real}(t)$, para ambos controladores y ambas variables de control (4 gráficos)

- Gráfico con trayectoria seguida por el robot en el plano bidimensional para controlados P y PI.

- Error promedio entre punto de partida y llegada (vértice inferior izquierdo del cuadrado), considerando 3 mediciones de error para un solo cuadrado. Presentar error para controlador P y PI.

En su presentación, compare de forma cualitativa las trayectorias seguidas por el robot en el plano 2D, con y sin control.

Incluir en paquete: archivo launch de nombre *avanzar_y_rotar_ctrl.launch* que permita iniciar tanto los nodos que implementan su rutina, como los nodos que implementan el simulador. Deben garantizar que sea posible correr el experimento completo simplemente ejecutando:

$$roslaunch < nombre\_paquete > avanzar\_y\_rotar\_ctrl.launch$$

Demostración: Movimiento en cuadrado controlado.

2. Comportamientos reactivos usando control P

Una de las ventajas de usar control automático es que comportamientos complejos pueden ser obtenidos con funciones simples. Los robots en general están dotados de variados comportamientos reactivos, los cuales no son previamente planeados sino que responden (o reaccionan) a estímulos externos. En la presente sección, utilizaremos un lazo de control para dotar a nuestro robot con la capacidad de moverse por un pasillo sinuoso como el mostrado en la figura 2, sin chocar con las paredes.

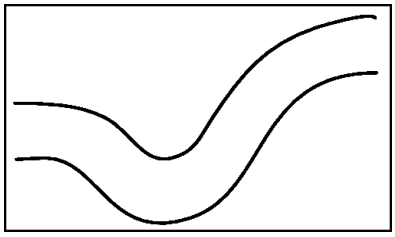

Figura 2: Pasillo sinuoso para navegación reactiva.

Mediante un controlador P, TurtleBot debe ser capaz de avanzar con una velocidad fija (0.2 [m/s]) a través de un pasillo de aproximadamente 0.8 [m] de ancho, y manteniendo una trayectoria equidistante a las paredes. Para esto debe detectar la distancia a las dos paredes del pasillo y calcular la diferencia entre ellas. Esta variable será nuestra señal a controlar por medio de la actuación sobre la velocidad angular del robot. Según esto, la señal de referencia deberá ser constante e igual a cero (diferencia de distancia a paredes igual a cero). Considere en su diseño una solución para los casos donde no será posible ver ambas paredes (curvas pronunciadas).En la presentación deberá explicar brevemente que criterio utilizó para obtener la distancia a ambas paredes.

Tanto para sus pruebas como para la demostración, contará con el mapa mostrado en la figura 2, el cual está compuesto por dos archivos: `pasillo.yaml` y `pasillo.pgm`.

El archivo `pasillo.yaml`, contiene toda la metadata asociada al mapa, y que permite definir parámetros como: la coordenada métrica que corresponde al pixel (0, 0) según la posición del sistema de coordenadas definido, el archivo de imagen que contiene el mapa, la resolución de la imágen que permite convertir pixels en metros, etc. A continuación se aprecia el contenido del archivo en cuestión:

```
image: pasillo.pgm
negate: 0
free_thresh: 0.196
occupied_thresh: 0.65
origin: [0.0, 2.96, 0.0]
resolution: 0.01
```

Por otra parte, el archivo `pasillo.pgm` contendrá la imagen que define la geometría del espacio. El contenido de este archivo luce exactamente igual que la figura 2.

Para cargar el mapa en el simulador, simplemente acceda al menú `File ->Open Map`, y seleccione el archivo `pasillo.yaml`.

Incluir en paquete: archivo launch de nombre *navegacion_pasillo.launch* que permita iniciar tanto los nodos que implementan su rutina, como los nodos que implementan el simulador. Deben garantizar que sea posible correr el experimento completo simplemente ejecutando:

$roslaunch < nombre\_paquete > navegacion\_pasillo.launch$

Demostración: Seguimiento de pasillo controlado.

3. Seguimiento de trayectorias

Una de las tareas más importantes en robótica móvil es el planeamiento de rutas. Esta es una tarea que consiste en un proceso deliberativo, mediante el cual, un robot puede definir una trayectoria dentro de un mapa. Mediante dicha trayectoria, el robot podrá alcanzar una posición objetivo sin colisionar con obstáculos fijos, como por ejemplo, paredes, muebles, etc. Una vez que la trayectoria fue planificada, ésta es obtenida como una secuencia de coordenadas (en nuestro caso 2D: (x,y)) que guiarán al robot hasta su posición de destino.

Tal como vimos en clases, para poder ejecutar una trayectoria contamos con el método *Follow The Carrot*. Este métod utiliza un controlador de la familia PID para poder seguir un punto distante (*carrot*), que se va moviendo sobre la trayectoria a una distancia fija del punto más cercano al robot.

En esta sección, usted deberá implementar el método *Follow The Carrot* utilizando un controlador de tipo P, PI o PID, hasta encontrar el que permite seguir la trayectoria de mejor forma posible. Para esta experiencia, usted contará con tres archivos de texto que contienen las coordenadas que definen las trayectorias de la figura 3. Estas coordenadas representarán los puntos (x, y) del espacio en el simulador, en el sistema métrico.

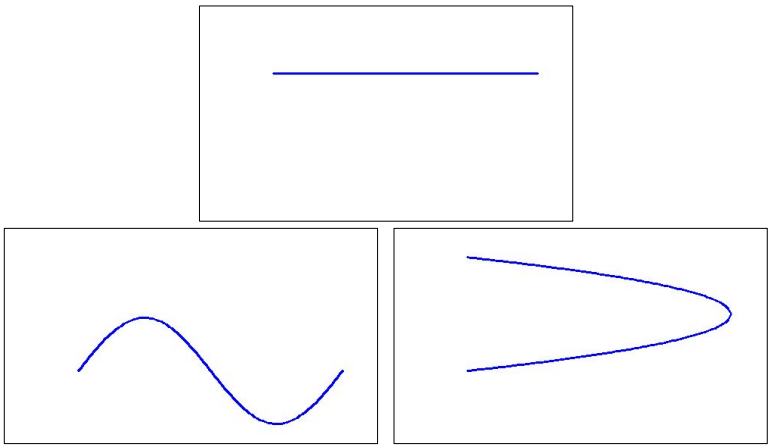

Figura 3: Trayectorias planeadas para seguimiento.

Para la implementación del método *Follow The Carrot*, deberá diseñar un controlador de la familia PID que controlará la orientación del robot, mientras aplica una velocidad lineal fija de 0.1 [m/s]. Este controlador recibirá como entrada: la orientación actual del robot (salida de la planta: $y(t)$) y el ángulo entre el centro del robot y la *zanahoria* (señal de referencia: $r(t)$). En base a estas entradas, el controlador enviará una velocidad rotacional a modo de actuación ($u(t)$), que intentará llevar a cero la diferencia entre los ángulos enviados.

Para interactuar con el controlador, implemente el nodo `follow_the_carrot` que deberá calcular el ángulo actual del robot y el ángulo formado entre el robot y la *zanahoria*, para luego enviarlos al controlador. Para el primer ángulo, use la información proveniente de la odomatría del robot. Para el segundo ángulo, usted primero deberá enviar la trayectoria hacia el nodo `follow_the_carrot` a través del tópico `nav_plan`, utilizando un mensaje de tipo `nav_msgs/Path` [2]. Para ello, implemente un cliente que lea cada archivo de texto con las coordenadas que definen la trayectoria, y encapsúlelo en el mensaje `nav_msgs/Path` para su envío. Una vez que el nodo `follow_the_carrot` cuente con la trayectoria, calcule el ángulo que forma el centro del robot con el punto de la trayectoria donde se encuentra la *zanahoria*, y envíelo como entrada al controlador.

Incluir en presentación: Gráfico con la trayectoria lograda versus la trayectoria original. Calcule el error cuadrático entre las curvas para las 3 trayectorias, y repórtelo en sus slides.

Incluir en paquete: archivo launch de nombre *follow_the_carrot.launch* que permita iniciar tanto los nodos que implementan su rutina, como los nodos que implementan el simulador. Deben garantizar que sea posible correr el experimento completo simplemente ejecutando:

$roslaunch < nombre\_paquete > follow\_the\_carrot.launch$

Demostración: Seguimiento de ruta para las tres secuencias entregadas. Durante la ejecución de las trayectorias, deberá levantar una ventana adicional donde se vea la trayectoria a seguir, y se aprecie la trayectoria real seguida por el robot en la medida que va avanzando.

Referencias

[1] http://wiki.ros.org/pid

[2] http://docs.ros.org/en/diamondback/api/nav_msgs/html/msg/Path.html