DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Técnicas de los Sistemas Inteligentes. Curso 2015-16. Práctica 1: Robótica

Entrega 2: Ejercicios prácticos para modificar el cliente/servidor

Objetivo

El objetivo de los siguientes ejercicios es servir como guía a la implementación de un gestor de goals para detectar un atasco en el comportamiento del robot y ayudar a un robot a salir de un atasco mediante la cancelación y envío de goals, en un proceso de navegación local. Estos ejercicios pueden hacerse de forma independiente al uso de costmaps, pero es recomendable integrar finalmente esta implementación con la llevada a cabo en los ejercicios de uso de costmaps.

Ejercicios

1. Gestionar mensajes de feedback

Comprobar que desde el servidor se envían mensajes de feedback en cada iteración del bucle de control (función ejecutaCB). Una vez comprobado esto se pide lo siguiente:

1.a) Definir en el cliente las funciones callbak apropiadas para gestionar información sobre activación de goal, realización de goal y feedback con las siguientes cabeceras:

```
void feedbackCBGoal0(const move_base_msgs::MoveBaseFeedbackConstPtr
&feedback);
```

```
void doneCBGoal0(const actionlib::SimpleClientGoalState& estado, const
move_base_msgs::MoveBaseResultConstPtr &resultado);

void activeCBGoal0()
```

1.b) Modificar la sentencia actual para envío de goals desde el cliente adecuadamente. Para ello consultar la api del cliente de actionlib http://docs.ros.org/jade/api/actionlib/html/classactionlib_1_1SimpleActionClient.html para saber qué parámetros opcionales admite la función sendGoals() y cómo usarlos para especificar qué funciones callback monitorizarán el seguimiento de un goal enviado.Consultar también, para información sobre cómo usar varios callbacks para un goal el tutorial http://wiki.ros.org/actionlib_tutorials/Tutorials/Writing%20a%20Callback%20Based%20Simple%20Action%20Client.

1.c) En la función activeCB (definida para cuando un goal está activo) simplemente mostrar un mensaje "El goal actual está activo".

1.d) En la función doneCB (definida para cuando un goal se ha realizado) usar variables booleanas para distinguir entre goal conseguido con éxito, goal abortado y goal cancelado. Para ello usar el valor del parámetro estado (definido en la cabecera de la función más arriba), que es asignado automáticamente por el servidor (ahora mismo no importa cómo, más tarde será necesario saberlo), y consultar en http://docs.ros.org/fuerte/api/actionlib_msgs/html/msg/GoalStatus.html todos los posibles valores que puede tomar.

1.e) Además, se requiere que la función callback para feedback, previamente implementada, capture en una variable la posición inicial recibida desde el servidor y calcule la distancia desde la posición actual. Consultar información sobre la estructra de mensajes de feedback (y resto de mensajes de move_base_msgs) en http://docs.ros.org/fuerte/api/move_base_msgs/html/index-msg.html. Mostrar información en pantalla sobre la posición actual capturada (mostrarlo en pantalla solo una vez) y sobre la distancia (mostrar la información cada segundo, usar ROS_INFO_THROTTLE). Ver información detallada sobre mensajes en la consola ros en http://wiki.ros.org/rosconsole.

2. Cancelar un goal desde el cliente

Comprobar que en el **servidor** hay una función callback (preemptCB) que se dispara cuando hay una petición de cancelación de goal desde el cliente. Esta función pone el estado del servidor (y del goal en curso) en PREEMPTED (con la función as.setPreempted()) cuando desde el cliente se solicita una cancelación de goal. Consultar la api del servidor de actionlib para más información sobre las funciones del server en http://docs.ros.org/jade/api/actionlib/html/classactionlib_1_1SimpleActionServer.html.

Observar también que en el cliente ya se ha implementado (en el ejercicio anterior) en la función doneCBGoal0(...) la gestión necesaria para capturar el estado actual del goal. Esta función se dispara cuando un goal es alcanzado con éxito, cuando es abortado o cuando es cancelado. Tener en cuenta que este estado del goal es asignado programáticamente en el servidor, de las siguientes formas:

- En la función preemptCB se usa la sentencia as. setPreempted() para poner el autómata interno del servidor en estado PREEMPTED (consultar información sobre los estados y transiciones del autómata interno del servidor en http://wiki.ros.org/actionlib/DetailedDescription#Simple_Action_Client

- En las últimas líneas de la función ejecutaCB del servidor se asigna el estado a SUCCEEDED o ABORTED dependiendo del éxito o no de consecución del objetivo. (as.setSucceeded y as.setAborted).

Una vez comprobado esto se pide lo siguiente **en el cliente** (hacerlo incrementalmente sobre lo hecho en el ejercicio anterior):

2.a) Modificar la función de feedback para que active una variable booleana que refleje la condición de cancelación: cuando la distancia recorrida por el robot sea mayor de, por ejemplo, 1 metro.

2.b) En la función main del cliente, después de enviar el goal, esperar mientras no haya condición de cancelación. Cuando haya condición de cancelación (distancia mayor de 1 metro) enviar un mensaje de cancelación de goal (con ac.cancelGoal(), ver api del cliente actionlib). Una vez enviada la cancelación, esperar un máximo de 10 segundos (ver ejemplo de uso de la función waitForResult(...) en el tutorial http://wiki.ros.org/actionlib_tutorials/Tutorials/SimpleActionClient%28Threaded%29) e

informar por pantalla si el estado del goal es cancelado o si ha habido algún problema (el estado del goal no es cancelado).

2.c) Observar también que el robot se detiene, esto es así porque el bucle de control del robot (while(true) en la función ejecutaCB del servidor) comienza con una condición en la que se comprueba que, si se ha recibido una petición de cancelación (as.isPreemptRequested()), la función ejecutaCB finaliza.

3. Detectar que el robot se ha atascado

Hay varias técnicas para detectar si un robot está atascado. En este ejercicio se pide implementar una simple modificando la función feedbackCB en el cliente (que según está programado en el servidor recibe mensajes de feedback cada 0.2 segundos, porque hemos definido su "rate" a 5 Hz en la sentencia ros::Rate rate(5)). La modificación consiste en comprobar si estamos en la misma posición actual durante más de un tiempo razonable. Pasado ese tiempo lo notificamos en una variable booleana (podemos usar la variable "cancelar" usada en el ejercicio anterior).

Consultar http://wiki.ros.org/roscpp/Overview/Time para usar variables de tiempo en ROS. Una mejora de este ejercicio puede ser considerar que la posición se mantiene en una zona circular de radio pequeño más de un tiempo razonable.

4.Enviar un nuevo goal.

Una vez detectado que el robot se ha atascado, podemos intentar sacarlo del atasco enviándole un nuevo goal. En este ejercicio basta con proponer una posición fija que lo aleje lo suficiente del atasco y comprobar que llega a esa nueva posición (en el siguiente ejercicio programaremos que retome el anterior goal para seguir hasta su objetivo principal). Para enviar un nuevo goal hay que tener en cuenta que

1. tenemos que definir en el cliente funciones callback distintas a las anteriores para gestionar la monitorización del nuevo goal (pensar si es necesario definir todas nuevas o podemos reutilizar alguna de las ya definidas).
2. hay que comprobar que el progreso del nuevo goal se realiza adecuadamente o no (e.d. si este nuevo goal acaba con éxito, abortado o cancelado) y proceder de acuerdo a ello.

5. Implementar una política de cancelación y reanudación de goal

Sobre la implementación resultante de los anteriores ejercicios, aplicar una política que permita al robot retomar el primer goal que se canceló por el atasco. La política de reanudación del goal puede ser la siguiente (a grandes rasgos):

1. Enviar goal de usuario (goal principal)
2. Si hay atasco
    a. Enviar otro goal secundario
    b. comprobar que se ha alcanzado el goal
    c. Si no se ha alcanzado el goal devolver error fatal y acabar (porque tratando de salir de un atasco ha habido algún problema).
    d. Si hay exito en el goal secundario
        i. Enviar goal principal otra vez
        ii. Esperar a que esté activo
        iii. Volver a 2.