Plataformas Software en Robótica.
Boletín 4. Acciones con ROS en C++

- **Ej1**. Vamos a desarrollar el ejemplo que tenéis explicado en los videos demostrativos y para ello vamos a crear un mensaje personalizado, un cliente y un servidor de acciones. El contexto es el siguiente: nuestro servidor de acciones va a recibir un mensaje con un campo en el que se le indica un número de segundos en los que va a estar "trabajando". El servidor simulará trabajar ese intervalo de tiempo e irá mandando un *feedback* con los segundos que lleva trabajando. Al finalizar, el servidor mandará al cliente un *result* con el número de segundos que estuvo trabajando hasta que la acción terminó (es posible cancelarla antes de tiempo) y un mensaje informando si pudo finalizar correctamente o si cancelaron el servicio de forma anticipada.
  - **Ej1.a**. Cread un mensaje personalizado de acción:
    - Goal
      - segundos (int32)
    - Result
      - total (int32)
      - mensaje (string)
    - Feedback
      - parcial (int32)
  - **Ej1.b.** Cread un servidor de acciones (estructurado). El servidor tendrá un callback para gestionar nuestros mensajes personalizados en el que simulará trabajar (un bucle con un *sleep_rate*) el número de segundos que le llega en el mensaje. Es necesario que el servidor sepa gestionar una cancelación anticipada.
  - **Ej1.c**: Cread un cliente de acciones que pida por pantalla un número entero que sirva para cubrir el mensaje personalizado (campo *segundos*) y, posteriormente, llamad al servidor de acciones con dicho mensaje.
    - El cliente deberá de esperar a que el servidor esté levantado para llamarlo
    - El cliente continuará trabajando mientras espera el resultado del servidor. Para esto tendrá que comprobar cada cierto tiempo (bucle) el estado de la acción solicitada.
    - El cliente tendrá implementado los diferentes callbacks para gestionar los estados de la acción (*feedback*, *done*, *active*).
  - **Ej1.d.** Modificad el cliente para que pida al usuario un segundo número que, en caso de alcanzarse en el feedback, sirva para cancelar la acción. Ej. segundos_trabajo=10, segundos_cancelar=5. Al llegar a 5 en el feedback se cancela la acción. Si el número es mayor que el de trabajo, la acción finalizará con normalidad.
- **Ej2.** Descargad el código fuente del servidor de acciones de Fibonacci (se encuentra en el aula virtual, fibonacci_action_server.cpp). Tenéis el código fuente pero necesitáis crearle un paquete **con las dependencias roscpp, actionlib y actionlib_tutorials**, cambiar el CmakeList.txt y el package.xml, y, finalmente, compilar y ejecutar (si se quiere lanzar con el *roslaunch* entonces también es necesario crear el directorio y el fichero correspondiente).
  - **EJ2.1**. Desarrollad un cliente para el servidor de acciones Fibonacci. **Necesitareis la dependencia *actionlib_tutorials*** por el mensaje *FibonacciAction*, que es el que usaremos para llamar a la acción.
    - Tenéis que consultar el mensaje *FibonacciAction* para saber como usarlo.
    - El cliente esperará a que el servidor esté levantado
    - Llamad a la acción a través de *waitForResult* y darle un número de segundos para esperar a que finalice antes del time out (ros::Duration(30))

        - Mostrad al final de la acción los valores de la secuencia obtenidos (para recorrer un vector podéis usar un iterador std::vector<int>::const_iterator) o podéis usar un bucle for entre 0 y vector.size()-1)
        - Cancelación de la acción: el goal del mensaje lleva el número de elementos de Fibonacci que se van a generar. El feedback nos devolverá la lista temporal generada. Si el último elemento de la lista es mayor que el numero de elementos solicitados, se cancelará la acción. Ej: numeros_solicitados=10, lista_feedback: 1,1, 2,3,5,8,**13** (en este punto se cancela)
- **Ej3**. Movimiento del robot (simulador turtlebot_stage) con acciones
    - **Ej3.1.** Cread un mensaje personalizado de acciones con los siguientes campos:
        - Goal: entero (tiempo)
        - Goal: float (velocidad)
        - Result: entero (tiempo efectivo)
        - Feedback: float (distancia central del robot al objeto más proximo)
    - **Ej3.2.** Cread un servidor de acciones que reciba el mensaje personalizado y haga mover el robot hacia adelante una serie de segundos (tiempo) a una velocidad dada (velocidad). El robot comprobará la distancia al obstáculo más cercano (escaner) y la devolverá en el feedback. La acción puede ser cancelada por el cliente. El servidor devolverá como resultado de la acción los segundos reales en los que se estuvo moviendo el robot.
    - **Ej3.3.** Cread un cliente que cubra la información del mensaje personalizado pidiendo los datos por terminal. El cliente invocará la acción y deberá cancelarla si la distancia a un obstáculo (información recibida por el feedback) es menor de 1 metro. El cliente al finalizar la acción mostrará su estado final y los segundos efectivos que el robot se estuvo moviendo

- **Ej4**. **Examen de enero de 2021** (adaptación)**: Generador de números aleatorios**
    - **Ej4.1.** Cread un mensaje personalizado de acciones con los siguientes campos
        - Goal
            - seed (int): semilla para inicializar un generador de números aleatorios
            - iterations (int): numero de iteraciones generando números aleatorios
        - Result
            - last_number (int): último número aleatorio generado en el bucle de la acción
        - Feedback
            - partial_number (int): número aleatorio generado en cada iteración
    - **Ej4.2.** Cread un servidor que genera números aleatorios en base al mensaje personalizado
        - El servidor tiene que gestionar las cancelaciones
        - El servidor devolverá en el *feedback* el elemento aleatorio creado en esa iteración
        - El servidor devolverá en el *result* el último elemento aleatorio generado antes de finalizar la acción
        - Para crear los números aleatorios podéis usar *srand/rand* de *stdlib*

```
#include <stdlib.h>//rand y srand
srand(seed);//establece la semilla para los números
rand()%10;//genera números aleatorios entre 0 y RAND_MAX por cada
          // llamada. El módulo nos permite gestionar el rango
```

    - **Ej4.3.** Cread un cliente que llame al servidor de acciones aleatorio. El cliente pedirá por pantalla los campos del mensaje personalizado: una semilla (*seed*) y un numero de iteraciones. Además pedirá por pantalla un número entero (*suma_maxima)* que

gestionará la cancelación de la acción según la siguiente lógica: el cliente irá sumando los números aleatorios recibidos a través del *feedback*. En el momento en que la suma de dichos números supere la *suma_maxima*, la acción será cancelada. Si nunca se llega a ese número, entonces la acción terminará con normalidad.

- **Ej4.4.** Modificad el servidor y el cliente desarrollados en los apartados anteriores
  - Servidor de acciones: añadidle un subscriptor a un topic llamado *topic_random_server* que mostrará por pantalla todos los mensajes recibidos (mensajes Int32).
  - *Cliente*: publicará el último número aleatorio recibido por la acción (se cancele o se finalice correctamente) a través del tópic: *topic_random_server* (mensajes Int32)