Guía kobuki\_multi\_robot

Diciembre, 2021

# Resumen

El objetivo de este documente es describir el paquete de ROS kobuki\_multi\_robot. Este paquete ha sido desarrollado con la finalidad de poder realizar experimentos en los que uno o más kobukis naveguen de forma autónoma recibiendo (desde una estación base) puntos objetivo a los cuales tendrán que dirigirse.

# 1. Descripción del experimento

El experimento consiste en publicar objetivos globales para N kobukis situados en el entorno y que estos sean capaces de navegar de forma autónoma y simultánea evitando colisiones con el resto de los kobukis.

El paquete kobuki\_multi\_robot permite poder realizar este experimento tanto en un entorno real como en un entorno simulado en Gazebo. Este paquete está disponible en un repositorio de GitHub y dos ejemplos de experimentos en simulación pueden encontrarse en:

<https://github.com/esauortiz/kobuki_multi_robot/blob/master/documentation/pictures/mission_default.gif>

<https://github.com/esauortiz/kobuki_multi_robot/blob/master/documentation/pictures/mission_willowgarage.gif>

# 2. Experimentos en entornos reales

## 2.1. *Base station*

La estación base o *base station* es utilizada para **monitorizar** el experimento, **publicar un mapa** único del entorno en el que se encuentran los kobukis y **publicar** **objetivos globales** sobre este mapa.

roslaunch kobuki\_multi\_robot base\_station.launch <global\_goals\_file\_name> <map\_name>

Dentro de base\_station.launch se incluyen los siguientes archivos *launch*:

* **rviz** con un fichero de configuración .rviz
* **map server** publicando el mapa <map\_name>
* **global goals publisher** que publica los objetivos globales incluidos en el fichero <global\_goals\_file\_name>

## 2.2. Conexión con la NUC montada sobre un kobuki

La conexión con la NUC montada sobre el kobuki desde la *base station* se realiza mediante **ssh**.

ssh username@ip\_address

En el paquete kobuki\_multi\_robot se proveen archivos .sh con los nombres de usuario y direcciones IP de cada uno de los kobukis, de tal manera que se puede iniciar la comunicación ejecutando el siguiente comando:

./<nombre\_del\_kobuki>\_login.sh

La **contraseña** de usuario en todas las **NUC** es 12345678.

Una vez establecida la conexión será necesario **acceder** a la carpeta del paquete kobuki\_multi\_robot,

roscd kobuki\_multi\_robot

**verificar** el **estado** del repositorio

git status

y **sincronizar** el repositorio si es necesario

git pull

## 2.3. Puesta en marcha de un kobuki

Para poner en marcha un kobuki es necesario ejecutar los **drivers ROS** de un kobuki y los **drivers del sensor** acoplado a la estructura del kobuki. Para tal fin, se provee de un archivo .sh que ejecuta un archivo launch configurando los argumentos con los valores correspondientes para cada kobuki. De esta forma, un kobuki puede ponerse en marcha ejecutando el siguiente comando desde un terminal (**abierto en la NUC de un kobuki**) en la carpeta /kobuki\_multi\_robot/start\_and\_nav:

./kobuki\_start.sh

Como se ha mencionado, este .sh ejecuta el archivo kobuki.launch pasando por argumentos el nombre KOBUKI\_NAME y el sensor SENSOR\_ATTACHED acoplado al robot, ambos definidos en el archivo ~/.bashrc de cada NUC.

Dentro de kobuki.launch se incluyen los siguientes archivos *launch*:

* Drivers ROS del kobuki (minimal.launch)
* Drivers ROS del sensor acoplado al kobuki (kobuki\_<SENSOR\_ATTACHED>.launch)
* TFs de kobuki (kobuki\_description.launch)

## 2.4. Puesta en marcha del sistema de navegación

El sistema de navegación se basa en el **paquete navigation de ROS** e incorpora un **nodo** que comprueba si hay algún **objetivo global** para el kobuki. Este sistema puede ejecutarse con el siguiente comando desde un terminal (**abierto en la NUC de un kobuki**) en la carpeta /kobuki\_multi\_robot/start\_and\_nav:

./kobuki\_navigation -m <mission\_label>

Donde mission\_label indica en qué posición se encuentra el kobuki. Dicha posición se establece en el archivo <KOBUKI\_NAME>.yaml ubicado dentro del directorio /kobuki\_multi\_robot/params/missions/<mission\_label>. El nombre de la carpeta missions está pendiente de cambiar por initial\_poses y el nombre del argumento mission\_label por initial\_pose\_label.

Este .sh ejecuta el archivo kobuki\_navigation.launch pasando por argumentos el nombre KOBUKI\_NAME, identificador KOBUKI\_ID y una etiqueta para establecer la posición inicial mission\_label del robot. Tanto KOBUKI\_NAME como KOBUKI\_ID están definidos en el archivo ~/.bashrc de cada NUC, de tal manera que únicamente será necesario indicar qué posición inicial debe tener cada kobuki.

Dentro de kobuki\_navigation.launch se incluyen los siguientes archivos *launch*:

* **AMCL** que recibe como argumento la posición inicial del robot (amcl.launch)
  + Como se ha mencionado, dicha posición se establece en el archivo /kobuki\_multi\_robot/params/missions/<mission\_label>/<KOBUKI\_NAME>.yaml
* Global y local **planner** (move\_base.launch)
* **Velocity smoother** (velocity\_smoother.launch)
* Nodo que **escucha objetivos globales** (multi\_robot\_slave.launch)
  + Además de leer objetivos globales y comprobar si el ID se corresponde con el del kobuki, llama al servicio de <KOBUKI\_NAME>/move\_base/clear\_costmaps

# 3. Experimentos en Gazebo

Además de los archivos launch descritos en las secciones anteriores, se proveen otros archivos launch adicionales en la carpeta /kobuki\_multi\_robot/simulation, junto a otros elementos necesarios para la simulación del entorno y kobukis en el simulador Gazebo.

## 3.1. *Base station*

La estación base o *base station* es utilizada para **monitorizar** el experimento, **publicar un mapa** único del entorno en el que se encuentran los kobukis y **publicar** **objetivos globales** sobre este mapa.

roslaunch kobuki\_multi\_robot base\_station.launch <global\_goals\_file\_name> <map\_name>

Dentro de base\_station.launch se incluyen los siguientes archivos *launch*:

* **rviz** con un fichero de configuración .rviz
* **map server** publicando el mapa <map\_name>
* **global goals publisher** que publica los objetivos globales incluidos en el fichero <global\_goals\_file\_name>

## 3.2. Gazebo

Tanto el entorno como los kobukis pueden ser simulados en el simulador Gazebo. Para tal fin, se puede ejecutar el siguiente archivo launch:

simulated\_playground.launch <world\_label>

Que recibe como argumento un nombre de los mundos disponibles en la carpeta /simulation/environment/worlds. La cantidad de kobukis que aparecerán en la simulación deberá de ser especificada en el archivo /simulation/spawn\_kobukis.launch.

## 3.4. Puesta en marcha del sistema de navegación

De manera similar a la puesta en marcha del sistema de navegación en un entorno real, el archivo launch principal es kobuki\_navigation.launch, poniendo en marcha (para cada kobuki) el nodo ACML, global planner, local planner, velocity smoother y nodo de escucha de objetivos globales. Sin embargo, este launch se incluye dentro del archivo simulated\_mission.launch, incorporando de manera adicional la acción de mover a cada kobuki a un determinado punto inicial. De esta forma, el sistema de navegación en simulación puede ponerse en marcha con el siguiente comando:

simulated\_mission.launch <mission\_label>

Donde <mission\_label> indica qué posición inicial se seleccionarán para cada kobuki. Dichas posiciones deben estar definidas en la carpeta /param/missions/<mission\_label>.

Como ya se ha mencionado tanto la carpeta missions como el argumento mission\_label están pendientes de cambio por initial\_poses e initial\_pose\_label, respectivamente. Por tanto, será necesario renombrar el argumento mission\_label que recibe simulated\_mission.launch, además de la carpeta missions de la que también depende este archivo *launch*.

NOTA: De igual forma que el archivo /simulation/spawn\_kobukis.launch puede ser modificado para configurar el número de kobukis simulados en gazebo, será necesario modificar el archivo simulated\_mission.launch para poner en marcha únicamente el sistema de navegación de los kobukis simulados.

# 4. Publicación de objetivos globales

Tanto en un entorno real como en un entorno simulado, la publicación de los objetivos globales se realiza mediante el siguiente comando:

rostopic pub /update\_global\_goals std\_msgs/Int8 "data: 1"

Nota: En el momento en el que se pone en marcha la *base station* será necesario especificar el nombre del archivo que contiene los objetivos globales <global\_goals\_file\_name>, ubicado en la carpeta /param/global\_goals. Cada línea de dicho fichero debe de tener el siguiente formato:

KOBUKI\_ID x y z roll pitch yaw

# 5. Otra información relevante

## 5.1. Definición de nuevas posiciones iniciales y objetivos globales

* Crear una nueva carpeta con posiciones iniciales en /param/missions
* Crear nuevo fichero de objetivos globales en /param/global\_goals
* Actualizar los paquetes de cada kobuki con git pull en la carpeta del paquete kobuki\_multi\_robot
  + Comprobar el status del repositorio con git status
  + Si hay algún archivo modificado no sincronizado con el repositorio y queremos sincronizarlo con el repositorio git checkout <archivo\_no\_sincronizado>

## 5.2. Crear un mapa del entorno

Es posible hacer un mapa del entorno con cualquiera de los kobukis.

Será necesario poner en marcha los drivers del kobuki y el sensor acoplado desde un terminal abierto en la NUC del kobuki, estando en la carpeta /kobuki\_multi\_robot/start\_and\_nav:

./kobuki\_start

Además, será necesario poder controlar el kobuki desde un teclado. Para tal fin, será necesario ejecutar los siguientes launch desde la *base station*:

roslaunch kobuki\_multi\_robot velocity\_smoother <robot\_name>

roslaunch kobuki\_multi\_robot keyop.launch <robot\_name>

El mapa se construye con el paquete *gmapping* y puede ser ejecutado con el siguiente comando:

roslaunch kobuki\_multi\_robot create\_map.launch <robot\_name>

Por último, será necesario guarda el mapa creado con el siguiente comando desde la carpeta donde se guardará el mapa:

rosrun map\_server map\_saver -f <map\_name >

## 5.3. *Group ns*

Es posible agrupar bajo un mismo namespace a un conjunto de nodos ejecutados desde un archivo launch.

Ejemplo:

<group ns=”namespace”>

<node>

</node>

</group>

De esta manera, tanto los nombres de los nodos como los tópicos asociados a estos tendrán como prefijo “namespace”. Es importante mencionar que este prefijo no se coloca delante de los parámetros que reciben. Para más información:

http://wiki.ros.org/roslaunch/XML/group

## 5.4. Documentación adicional

* Tutoriales kobuki ROS: http://wiki.ros.org/kobuki/Tutorials