Занятие № 7. Gazebo

Описание

Rviz не является единственным способом визуализировать информацию в ROS. Ещё одним способом отобразить визуально состояние системы является пакет Gazebo. Конечно, нет необходимости разрабатывать и пользоваться двумя пакетами, которые выполняют одинаковые функции, и говорить о том, что это два равноценных способа визуализации неверно. На самом деле Gazebo - это немного большее, чем просто визуализатор; это целый симулятор мира с описанной физикой. Кроме того, в gazebo основными объектами являются не точки и линии, а трёхмерные объекты, обычно более или менее чётко нарисованные (вместо условных точек и сфер в Rviz).

Например, один из самых простых объектов в Gazebo - это робот «пионер» Pioneer 2 DX. Так он выглядит в симуляторе Gaztbo.

Основная цель симулятора - демострировать результат работы программы в виде, удобном для восприятия. Для описания внешнего вида объектов используется язык разметки xml. Некоторые объекты уже описаны и могут быть скачаны с официального сайта gazebo. Однако, объекты можно создавать самостоятельно из простых фигур, типа кубов, сфер и прочих.

Установка и использование

Gazebo является отдельным пакетом, который может быть использован независимо от ROS. В этом случае необходимо писать «скрипты», управляющие поведением объектов в симуляторе. Однако, возможно использовать Gazebo и совместно с ROS.
Для того, чтобы начать работу, необходимо установить последнюю версию Gazebo.

```
curl -ssL http://get.gazebosim.org | sh
```

Такая команда установит самую полную версию Gazebo. Если необходима ручная настройка компонентов, можно воспользоваться step-by-step инструкцией на сайте официального разработчика http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install. Важно отметить, что модели роботов не будут скачаны из общего хранилища. Их необходимо скачивать отдельно. Интерфейс Gazebo позволяет это делать «на лету», то есть при первом обращении к объекту, если он не будет найден в системе, будет предпринята попытка скачать объект с таким именем из базы данных. После скачивания, текстуры и xml объекта будут располагаться в ~/.gazebo/models.

Для запуска Gazebo независимо от ROS достаточно в терминале вызвать

```
gazebo
```

При этом появится пустой мир, в который можно добавлять объекты, двигать их, задавать освещение и прочее. Для того, чтобы связать Gazebo и ROS, необходимо скачать дополнительные пакеты:

```
sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-
```

```
control
```

Теперь можно вызвать

```
roscore && rosrun gazebo_ros gazebo
```

или

```
roslaunch gazebo_ros empty_world.launch
```

Теперь можно увидеть, что gazebo запущена как самостоятельная нода

Объект в Gazebo можно создать «вручную», но это не удобно с точки зрения практического использования, поэтому существует сервис, позволяющий создать объект.

Разберём код на C++, помещающий в симулятор робота и передвигающий его в точку (2, 0, 0).

```
#include "ros/ros.h"
#include "gazebo_msgs/SpawnModel.h"
#include "gazebo_msgs/ModelState.h"
#include <fstream>
#include "string.h"

using namespace std;

int main(int argc, char** argv) {
    ros::init(argc, argv, "gaztest");
    ros::NodeHandle node;
    ros::service::waitForService("gazebo/spawn_sdf_model");
    ros::ServiceClient add_robot =
node.serviceClient<gazebo_msgs::SpawnModel>("gazebo/spawn_sdf_model");
    gazebo_msgs::SpawnModel srv;

    ifstream fin("/home/usr/.gazebo/models/pioneer2dx/model.sdf");

    string model;
    string buf;
    while(!fin.eof()){
        getline(fin, buf);
        model += buf + "\n";
    }
    srv.request.model_xml = model;
    srv.request.model_name = "robot";
    geometry_msgs::Pose pose;
    srv.request.initial_pose = pose;
    add_robot.call(srv);
    //Spawning finished

    ros::Publisher pub =
node.advertise<gazebo_msgs::ModelState>("gazebo/set_model_state", 10);
    sleep(1.0);
```

```
    gazebo_msgs::ModelState msg;
    msg.model_name = "robot";
    msg.pose.position.x = 2.0;
    pub.publish(msg);
    sleep(1.0);
    ros::spinOnce();
    return 0;
}
```

Первая половина программы посвящена вызову сервиса, создающего объект. Особое внимание следует уделить файлу, где описана структура робота. Для корректной работы в сервис необходимо передать не путь к файлу, описывающему робота, а само содержимое файла. На данный момент существует два стандарта описания роботов: .sdf и .urdf; первый является более новым и более удобным для использования, поэтому большинство объектов на данный момент существует именно в формате .sdf.

Следует обратить внимание, что файл model.sdf должен быть скачан до начала работы программы. Для этого можно просто открыть gazebo и попытаться поместить робота pioneer_2dx на плоскость. При этом некоторое время будет происходить скачивание дискриптивных файлов робота, после чего все требуемые файлы появлятся в стандартном каталоге.

После размещения робота в мире его нельзя подписать на какой-то топик, как это было сделано в rviz. Для взаимодействия со всеми объектами gazebo использует один топик **gazebo/set_model_state**. При этом в передаваемом сообщении указывается имя объекта, которому оно адресовано. Из этого следует, что одновременно в мире не может существовать двух объектов с одинаковым именем. Отличие gazebo от rviz в этом моменте заключается в том, что rviz в случае прихода сообщения с просьбой создания объекта с существующим идентификатором создаст новый объект и удалит старый, а gazebo проигнорирует просьбу о создании.

Управление положением объекта происходит через топик gazebo/set_model_state или через одноимённый сервис. Различие в управлении через топик или через сервис в типе передаваемого сообщения и в наличии или отстутствии подтверждения о приёме сообщения. Интерес представляет сообщение, которое передаётся в топик gazebo/set_model_state. оно имеет три поля:

- model_name – имя объекта
- pose – абсолютное положение робота
- twist – относительное изменение положения

Следует отметить, что, приняв сообщение с непустыми значениями pose и twist, робот будет использовать только положение pose; twist используется только как справочная информация, например, для лога.

Для того, чтобы представленный выше код заработал, необходимо в CMakeLists.txt добавить зависимость от gazebo:

```
find_package(catkin REQUIRED COMPONENTS
  gazebo_ros
)
```

```
catkin_package(
  DEPENDS gazebo_ros
)

include_directories(
  ${catkin_INCLUDE_DIRS}
  ${GAZEBO_INCLUDE_DIRS}
  ${SDFormat_INCLUDE_DIRS}
)
```

В package.xml также необходимо указать build depend и run depend от gazebo_ros.