Department of Electrical and Electronic Engineering

ELEN90066 Embedded System Design

Workshop Four (W04) - (Remote)

Simulating a robot in a virtual environment

Welcome to Workshop 4 for Embedded System Design. In this workshop, you will finally be getting around to spawning a Kobuki robot into a world in Gazebo. You will be controlling the robot without using its sensors (i.e. open-loop control) to ensure that everything is working before moving on to designing the control algorithm in future workshops. In this workshop you will have some fun with simulation by recording and playing back an epic robot sporting moment!

There are again two options for using the software in this workshop:

1. Locally on your own computer
2. Remotely via MyUniApps

See the Workshop 3 notes for further details.

Pre-workshop background

Pre-workshop questions

1. Quick refresher question from Workshop 3 - what does the `catkin_make` command do?
2. You will be building a new world for this Workshop - this time by directly editing the world file. Calculate the Gazebo pose values for the four jersey barriers in Figure 1. State your answers as a set of four, six-element vectors (e.g. `[0 0 0 0 0 0]`).

   Recall that the Gazebo system coordinate system is shown by the coloured axes in Figure 1b, with the positive x-axis represented by the red line, the positive y-axis represented by the green line and the positive z-axis represented by the blue line. By default, with no rotation, the barriers are aligned to be parallel to the x-axis.

   Hint: The barriers are 4m long, approximately 1m wide at the base, and the pose coordinates are referenced to the mid-point of the barrier.

Workshop Exercises

In this workshop you will do the following

- Import the Kobuki robot model into a ROS package and spawn it into a custom-built Gazebo world. (TASK 1 - 5 marks)
- Perform basic open-loop control of the Kobuki through publishing ROS messages. (TASK 2 - 10 marks)
- Use the record feature in Gazebo to record the Kobuki's interactions with the simulated environment and ROS bags to store the robot's sensor values during simulation. (TASK 3 - 10 marks)

There are 25 marks available for successful completion of all tasks in this workshop, worth 1% of your final mark in the subject.

Importing the Kobuki robot model

Follow the directions in Workshop 3 to create and build a catkin workspace called `ws_04` in a source folder called `ws04_ws/src` under the home directory (repeated here for convenience):

```
$ cd
$ mkdir -p ws04_ws/src
$ cd ws04_ws
$ catkin_make
```

and use the `catkin_create_pkg` command to create a ROS package called `ws04_gazebo` in the `src` folder:.

```
$ cd src
$ catkin_create_pkg ws04_gazebo
```

Check that your new package is registered with ROS:

```
$ rospack list
```

Also, note the `kobuki_description` ROS package in the list. The model for the Kobuki, including it's physical characteristics, dynamics and sensors is in this package, stored as a `.xacro` file. It is included as standard in the ROS kinetic installation. You will be spawning this model into your own world. But first, let's take a look at the robot model...

The `roscd` command allows you to change directory (cd) directly to a ROS package or a stack. Enter the following commands to enter the `kobuki_description` package folder:

```
$ roscd kobuki_description
```

Hint: As you are typing the package name you can also use the Tab key to autocomplete it.

Now print the working directory using the Unix command `pwd`:

```
$ pwd
```

For the ESD VM, you should see:

```
/opt/ros/kinetic/share/kobuki_description
```

Note that `roscd`, like other ROS tools, will only find ROS packages that are within the directories listed in your `ROS_PACKAGE_PATH`. To see what is in your `ROS_PACKAGE_PATH`, type:

```
$ echo $ROS_PACKAGE_PATH
```

Now, navigate to the `urdf` folder in the `kobuki_description` ROS package and open up the following urdf file using nano

```
$ nano kobuki.urdf.xacro
```

This is the model of the Kobuki robot. Have a look through some of the parameters – you might be able to decipher what some of them represent. It is in here that you could alter the physical properties of the model or add additional sensors. Let's leave it stock... for now.

Ok, now it's time to spawn this robot into a Gazebo world! You will do this in two parts:

- Creating a Gazebo world file in your `ws04_gazebo` package that includes elements of the environment (barriers, etc.) and sets some additional Gazebo options (e.g. physics simulator); and
- Creating a launch file in your `ws04_gazebo` package that spawns the robot and sets up ROS.

Creating the world file

Create a `worlds` folder within the `ws04_gazebo` package and then a `ws04.world` file in that folder with the following contents:

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <!-- turn off shadows to reduce computational cost -->
    <scene>
      <shadows> 0 </shadows>
    </scene>

    <!-- Custom physics settings to speed up simulation -->
    <physics type='ode'>
      <max_step_size> 0.01 </max_step_size>
      <real_time_factor> 1 </real_time_factor>
      <real_time_update_rate> 100 </real_time_update_rate>
      <gravity> 0 0 -9.8 </gravity>
    </physics>
  </world>
</sdf>
```

A bit of further explanation about this file format - the SDFormat (Simulation Description Format), sometimes abbreviated as SDF, is an XML format that describes objects and environments for robot simulators, visualisation, and control. Originally developed as part of the Gazebo robot simulator, SDFormat was designed with scientific robot applications in mind. Over the years, SDFormat has become a stable, robust, and extensible format capable of describing all aspects of robots, static and dynamic objects, lighting, terrain, and even physics. You can find examples of the format over at http://sdformat.org/spec?elem=sdf&ver=1.4

Creating the launch file

Similar to Workshop 3, create a `launch` folder within the `ws04_gazebo` package, and then create a `ws04.launch` file in that folder with the following contents:

```
<launch>
  <!-- We inherit the logic in empty_world.launch, changing only the name of
     the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find ws04_gazebo)/worlds/ws04.world"/>
    <!-- more default parameters can be changed here -->
  </include>

  <arg name="robot_name" value="mobile_base"/>

  <!-- First, convert the kobuki xacro description and put on parameter server
      -->
  <param name="robot_description"
  command="$(find xacro)/xacro.py '$(find kobuki_description)/urdf/
     kobuki_standalone.urdf.xacro'"/>

  <!-- Now, spawn the robot into Gazebo -->
  <node pkg="gazebo_ros" type="spawn_model" name="spawn_$(arg robot_name)"
      args="-x 0.0 -y 0.0 -z 0.0 -unpause -urdf -param robot_description -
         model $(arg robot_name)" respawn="false">
  </node>

  <!-- Calculate the forward kinematics of the robot and publish the results
     via tf -->
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="
     robot_state_publisher">
    <param name="publish_frequency" type="double" value="30.0" />
  </node>

  <node pkg="nodelet" type="nodelet" name="$(arg robot_name)_nodelet_manager"
     args="manager"/>

</launch>
```

This code does several things

1. Includes the world file you just created by searching for your `ws04_gazebo` package and

including the specific world file;

2. Converts the Kobuki xacro description to urdf and puts it on a ROS parameter server (to send to the `spawn_model` script in the next step);

3. Spawns the robot at the location (pose) `x=0`, `y=0`, `z=0` using the `spawn_model` script in the `gazebo_ros` package;

4. Calculates the forward kinematics of the robot and publishes the results via tf [1].

Note in order to spawn the robot with a different angular pose, you can use the additional arguments to the `spawn_model` script: `-R` (roll in radians), `-P` (pitch in radians), `-Y` (yaw in radians). If omitted, as in the code above, by default they are all set to 0.

TASK 1 Modify your `ws04.world` file to create the world shown in Figure 1 with four jersey barriers at the specified locations.

You will need to manually edit the world file in the nano text editor to spawn the barriers as models with poses. See the world file in Workshop 3 for an example of spawning a Gazebo model with a certain pose.

The jersey barrier model name is `jersey_barrier` and the pose coordinates are specified as `x y z roll pitch yaw`, where `roll pitch yaw` are in units of radians.

> **Tip:** You can quickly add models to a world in the Gazebo GUI, however if you then save the world via File →Save World As, it will create a new world file which will contain the state of the entire world at that specific moment (this is ALL the information that is needed to launch and continue the simulation from that point in time). Gazebo will have expanded and replaced all the include tags in the initial description with the model definitions, which can become rather messy.

Test that your world is correct by loading it directly in Gazebo (don't use `roslaunch` just yet!)

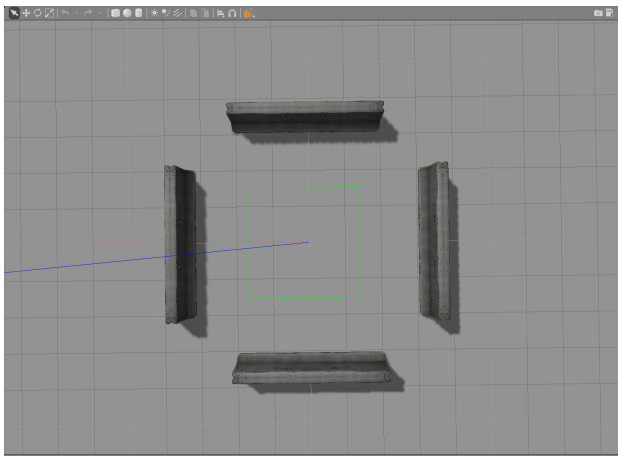
If all is well, spawn the Kobuki robot in your world by running:

```
. ~/ws04_ws/devel/setup.bash
roslaunch ws04_gazebo ws04.launch
```

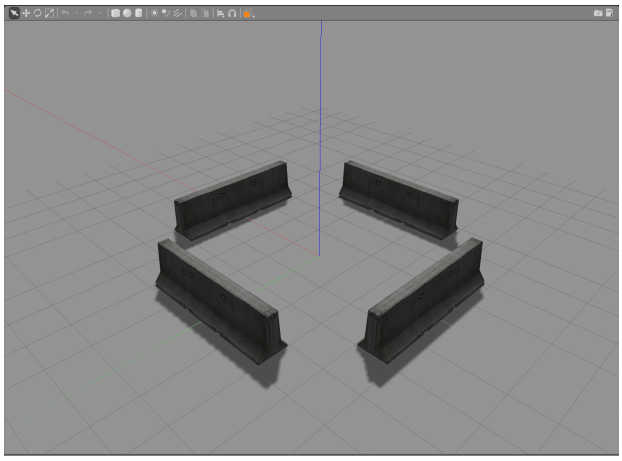
Show your demonstrator the successful launching of Gazebo with the correct world and the spawned Kobuki via the `roslaunch` command as above.

There are 5/25 marks for successful completion of this task.

[1] tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.



(a) Object positions for Task 1, viewed from above.



(b) Object positions for Task 1, viewed from front.

Figure 1: Object positions for Task 1

Controlling the Kobuki robot model

Remote controlling the Kobuki robot

Now that you can spawn a Kobuki robot model into a world, it's time to control it!

First, you will install a basic keyboard controller to allow you to directly control the Kobuki robot model and demonstrate the power of using ROS:

- **If you are using the ESD VM on your own computer:**

  Enter the following into a terminal window (you will need an internet connection to download the package(s)):

  ```
  $ sudo apt-get install ros-kinetic-kobuki-keyop
  ```

  and enter the password `embedded` when prompted and then `Y` to continue.

- **If you are using the ESD VM on MyUniApps:**

  Please see the video instructions on LMS to manually install this package as this VM has no direct internet access and will fail to download the packages.

This has just installed the `kobuki_keyop` package into the default ROS package folder. This ROS package contains an application that controls the Kobuki via keyboard commands. Try it out now:

1. In a Terminal window, use `roslaunch` and your launch file to spawn the Kobuki into the world from Task 1.

2. In a **new** terminal window, start the keyop application in the `kobuki_keyop` package by using the `keyop.launch` launch file

```
$ roslaunch kobuki_keyop keyop.launch
```

3. Read the on-screen instructions to provide keyboard commands to drive the Kobuki motors in the main Gazebo window. **Note:** You must keep the keyop terminal window in focus for it to capture the key strokes.

While you are having some fun with steering the Kobuki around inside your small 'robot arena', have a think about what the `keyop` application might be doing, e.g. using publish messages to control the Kobuki motors.

Controlling the Kobuki robot by publishing commands

A quick refresher of some of the ROS commands from Workshop 2. Open another new terminal window and use the `rostopic list` command to see the available ROS topics. The ones starting with `/mobile_base/` are the ones that relate to the Kobuki - either commands, sensors or events. Use the `rostopic info` command to print information about a particular topic, for example:

```
$rostopic info /mobile_base/commands/velocity
```

should output something like:

```
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
* /gazebo (http://esd-VirtualBox:43523/)
```

Use the `rostopic echo` command to print information about a particular topic to the screen, for example:

```
$rostopic echo /mobile_base/commands/velocity
```

You can publish your own commands to a topic by using the `rostopic pub` command. See here for the syntax of the `rostopic pub` command.

Now it's time to do some 'publishing' to control the Kobuki!

TASK 2 Close any terminal and/or Gazebo windows that you have open, open a new terminal and re-launch your setup from Task 1, spawning the Kobuki in the centre of the 'robot arena'.

Your task is to get the robot to drive in a circle of any radius inside the arena and stop on command (by disabling the motors) using only `rostopic pub` commands.

Hint #1: To get the Kobuki robot to continually move, you will need to use the `-r` rate option for the `rostopic pub` command for the `/mobile_base/commands/velocity` topic. With the rate mode, rostopic will publish your message at a specific rate. For example, `-r 10` will publish at 10Hz.

Hint #2: To get the Kobuki robot to stop by disabling the motors, you will need to publish to the `/mobile_base/commands/motor_power` topic using the correct topic type. See here for further details on the MotorPower Message definition.

Show your demonstrator the Kobuki driving in a circle and then successfully stopping by disabling the motors as described above.

There are 10/25 marks for successful completion of this task.

Recording simulation data

To observe both Gazebo (environment) and ROS (robot) topic data as the simulation is being run, you can use the `rqt` application.

To add a plot to `rqt`, go to:

Plugins→Visualization→Plot.

To easily add topics, open the Topic Monitor panel by going to:

Plugins→Topics→Topic Monitor

and check the boxes of the topics you wish to monitor. You can then drag them into the plot panel. For example, in Figure 2 the x and y coordinates of the robot position (reported by Gazebo, not the robot!) are plotted while it is doing a circle.

This is useful for 'on the fly' visualisation, however, if you wish to capture a set of data for post-simulation analysis or to replay it, there are several ways this can be done - by recording the simulation in Gazebo, or by storing ROS topic data in ROS *bags*.

You final task in this workshops is to record and replay simulation and sensor data as your robot completes a crucial task... robot soccer!

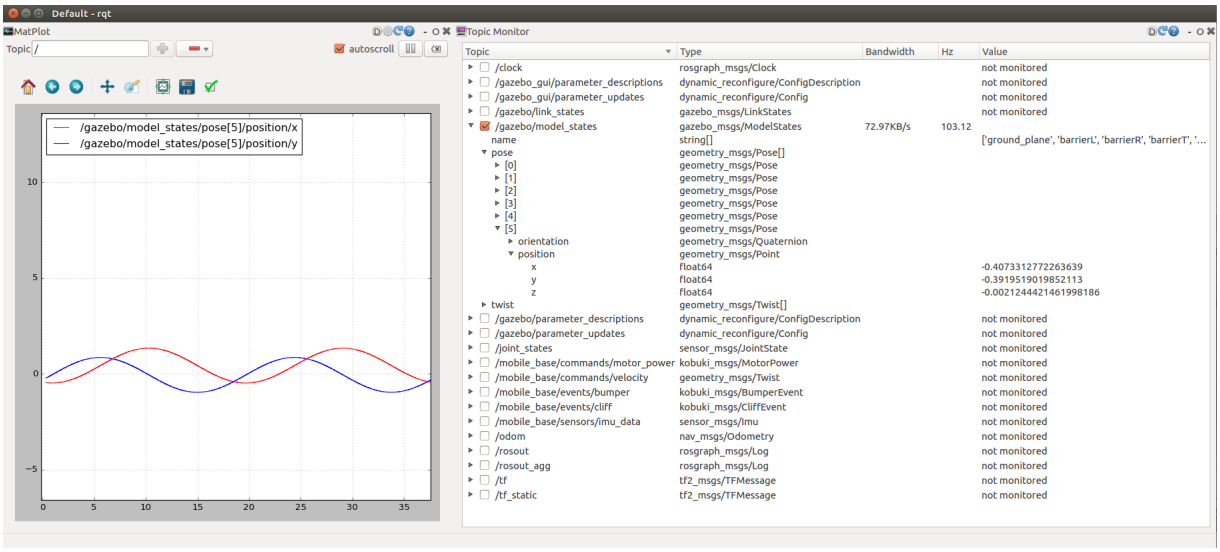


Figure 2: Plotting topic data in rqt

Recording and playing back simulation data in Gazebo

Have a read for a description of how Gazebo's logging capabilities can record your simulation and then reproduce it afterwards, using either the GUI or the command line.

Saving and playing back simulation data in ROS

Note: Before starting to save simulation data with the `rosbag` command, you must clear some disk space from the ESD VM as the command will not run unless you have more than 1GB of space available. Enter the following commands to free some space by removing all packages kept in the apt cache, regardless of age or need:

```
$sudo apt-get clean
```

Have a quick read through the documentation for the `rosbag` command, particularly for the `record` option. There's a LOT in there, so don't read everything, but it will be a handy guide for what is coming next.

To play back bag data, you can use the `rqt` command and go to the: Plugins → Logging → Bag menu item to open up a saved ROS bag.

TASK 3 To setup for this task, you must follow these steps in order:

1. Close any terminal and/or Gazebo windows that you have open to ensure no ROS cores or Gazebo instances are running.
2. Open a **new terminal** and re-launch your setup from Task 1, spawning the Kobuki in the centre of the 'robot arena'.
3. This time, use the Gazebo GUI to add a unit sphere at position (pose) `x=-1`, `y=-2`, `z=0.5`. This will be the ball. Don't worry, the Kobuki is small but pretty tough!

   Note: This time don't save over your world file. If you *really* want to save it, use "Save World As" and choose another file name.
4. Open a **new terminal** window and run the `kobuki_keyop` application ready to control the Kobuki.
5. Before you give any commands to the Kobuki, start recording the simulation in Gazebo using the instructions in the previous section.
6. Open a **new terminal** window and run the `rosbag` application to record ONLY the `/mobile_base/events/bumper` topic using the instructions in the previous section as a guide.

Now, for the actual task.... your task is to use the keyboard to control the robot to push the ball out of the 'arena', while recording the spectacular moment in Gazebo. You will need to replay this file to your demonstrator in order to get marked for this task.

What you must show your demonstrator:

1. A replay of the Gazebo simulation using the `gazebo -u -p` command showing the robot achieve glory by pushing the ball out of the arena;
2. A replay of the ROS bag file using `rqt` showing the bump sensor data during the 'game'.

There are 10/25 marks for successful completion of this task.