1. The importance of ROS for SDC

Bundle packages together, some package may crash. Using ROS will be helpful.

1. Overview of Catkin Workspace

Catkin is a powerful buil and package management system provided by ROS.

A catkin workspace is essentially a directory where Catkin package are build,modified and installed.Typically when you develop ROS based project you will be working out of a single workspace. This singular workspace will hold a wide variety of Catkin packages.All ROS software componets are organized into distributed as Catkin packages. Similar workspace, Catkin packages are nothing more than directories containing a variety of resources which when considered together constitute some sort of useful module. Catkin packages may contain source code for nodes,useful scripts,configuration files and more.In the section that follow you will see how to create a Catkin workspace,add packages to it,manage inner package dependencies,how to successfully compile.



**Catkin packages**

ROS software is organized and distributed into packages, which are directories that might contain source code for ROS nodes, libraries, datasets, and more. Each package also contains a file with build instructions - the CMakeLists.txt file - and a package.xml file with information about the package. Packages enable ROS users to organize useful functionality in a convenient and reusable format.

**Catkin workspaces**

A catkin workspace is a top-level directory where you build, install, and modify catkin packages. The workspace contains all of the packages for your project, along with several other directories for the catkin system to use when building executables and other targets from your source code.

1. Create a Workspace

All of the ROS related code you develop throughout this course will reside in your catkin workspace. You only need to create and initialize the workspace once.

Step 1: mkdir -p ~/catkin\_ws/src

Step 2: cd ~/catkin\_ws/src

Step 3: catkin\_init\_workspace

Step 4: cd ~/catkin\_ws

Step 5: catkin\_make

Step 6: Commentary

Congratulations! You just created your first catkin workspace.

Before you begin to work with and develop your own ROS package, you should take a moment to get acquainted with catkin workspace conventional directory structure as described in the ROS Enhancement Proposal (REP) 128: <http://www.ros.org/reps/rep-0128.html>

1. Add a package

**Cloning the simple\_arm Package**

One of the biggest benefits of using ROS is that it has a really large community of users and developers, so there is a lot of code that you can use.

Let’s clone an existing package and add it to our newly created workspace.

You will start by navigating to the src directory and cloning the simple\_arm package for this lesson from its github repo.

$ cd ~/catkin\_ws/src

$ git clone https://github.com/udacity/simple\_arm\_01.git simple\_arm

**Building the simple\_arm package**

After the repo has finished cloning, you can change directory to the top-level of the ros workspace and build the new package.

$ cd ~/catkin\_ws

$ catkin\_make

I see a CMake Error. "Could not find a package configuration file provided by controller\_manager"

**Installing Missing Packages Using apt-get**

I happen to know that controller\_manager refers to a ROS package from ROS Control. We can fix this by installing the associated Debian package. If I didn't already know this, I would probably have to rely on a Google search to figure out the exact name of the package required.

$ sudo apt-get install ros-kinetic-controller-manager

Some students have had success using the following commands to install missing packages:

$ source devel/setup.bash

$ rosdep install simple\_arm

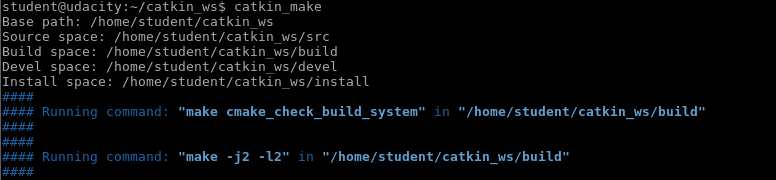
OK, now that we have the controller-manager package let’s try building again. I'm still in the top level directory, so I can just type “catkin\_make” and hit enter.

$ catkin\_make

Looks like the build worked. Great, that wasn't so bad. Let’s run some of this code that we just cloned!

**Trouble shooting:**

1. Using the VM provided by Udacity, after enter”catkin\_make” under the path of ~/catkin\_ws, there are no buiding information was popped up.



1. Roslaunch

roslaunch allows you to do the following

* Launch ROS Master and multiple nodes with one simple command
* Set default parameters on the parameter server
* Automatically re-spawn processes that have died

To use roslaunch, you must first make sure that your workspace has been built, and sourced:

$ cd ~/catkin\_ws

$ catkin\_make

Once the workspace has been built, you can source it’s setup script:

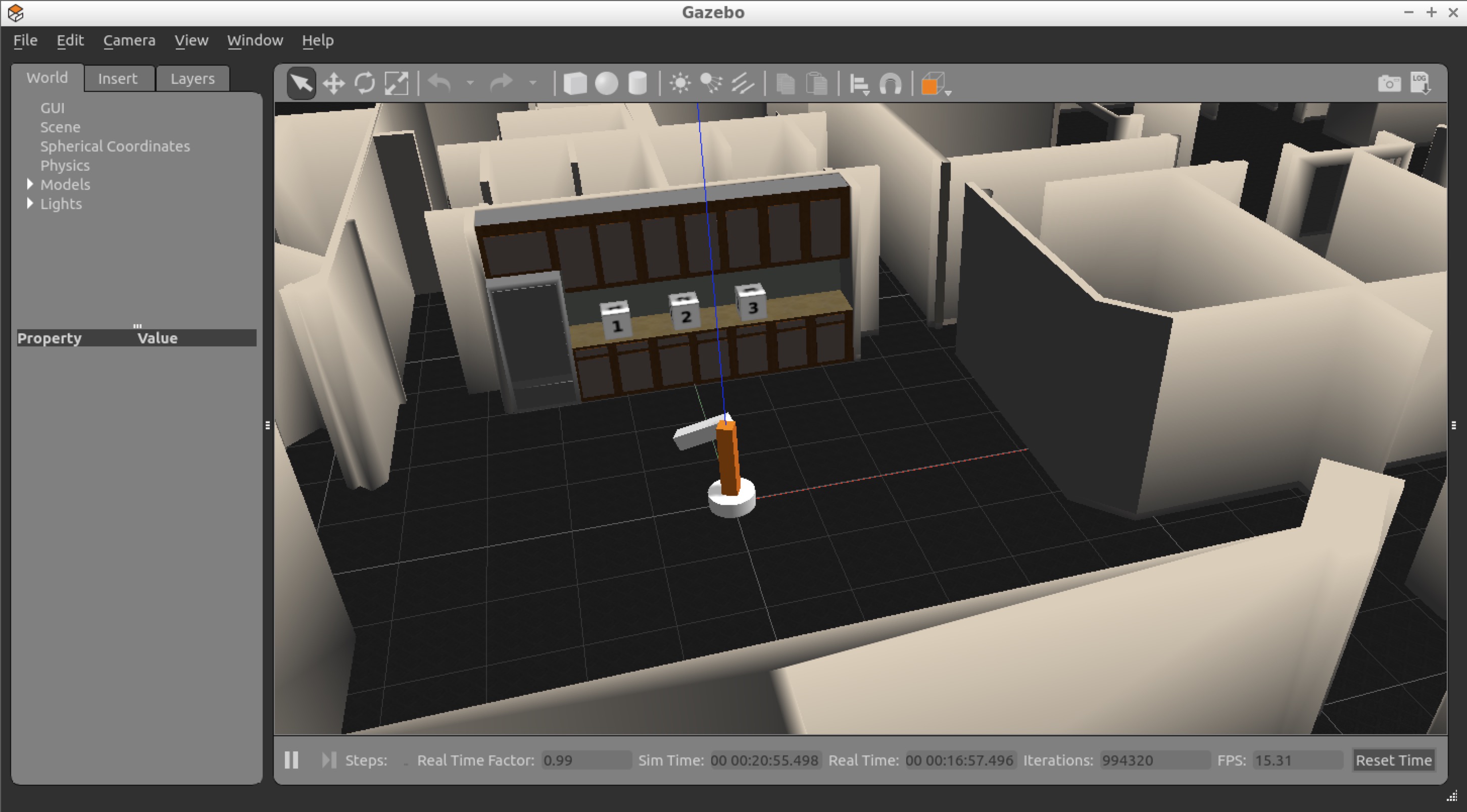
$ source devel/setup.bash

With your workspace sourced you can now launch simple\_arm:

$ roslaunch simple\_arm robot\_spawn.launch

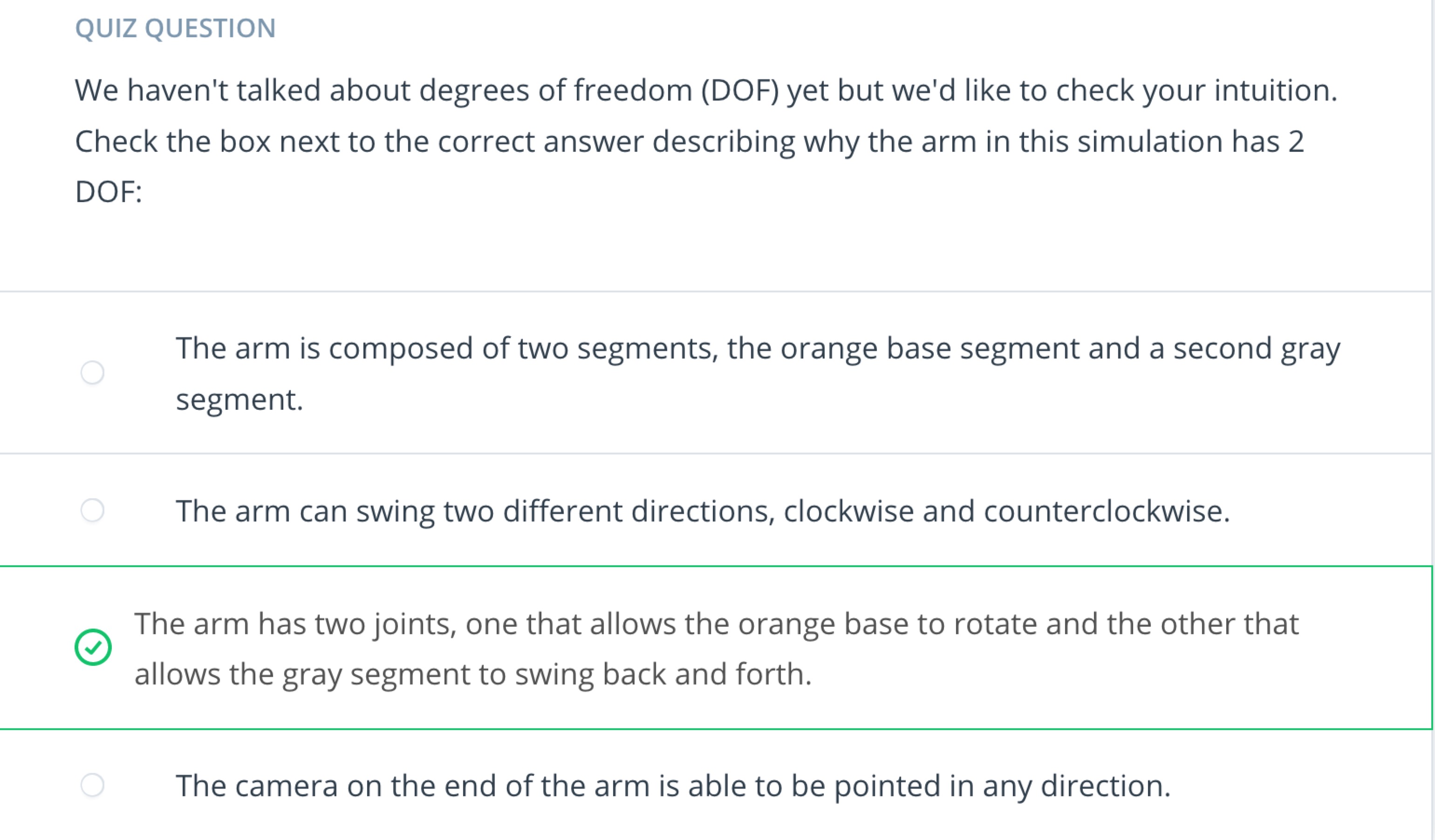
Finally, you will see the following picture which is showed out.

It may take some time for showing the result, just wait and wait if your computer is not that fast, during the waiting time nothing except grey or black back ground will show up.



And there you have it! Your very own two-degree-of-freedom arm in simulation!

**Hint:** To figure out why the arm is just swinging around loosely, check out the log messages in the ROS master console.



1. Rosdep

After the last exercise, you might have noticed the following warning line:

The controller spawner couldn’t find the expected controller\_manager ROS interface.

ROS packages have two different types of dependencies: build dependencies, and run dependencies. This error message was due to a missing runtime dependency.

The rosdep tool will check for a package's missing dependencies, download them, and install them.

To check for missing dependencies in the simple\_arm package:

$ rosdep check simple\_arm

**Note**: In order for the command to work, the workspace must be sourced.

This gives you a list of the system dependencies that are missing, and where to get them.

To have rosdep install packages, invoke the following command from the root of the catkin workspace

$ rosdep install -i simple\_arm

Issues with this command may arise when using a VM. If this is the case, please try:

sudo apt-get install ros-kinetic-gazebo-ros-control

And there you have it, launch the simple\_arm package again and your arm should no longer be swinging around!

With all packages properly installed, you will now learn more about ROS packages, in preparation for writing your own nodes in the next lesson!

1. Dive Deeper into packages

Here you'll begin your dive into ROS packages by creating one of your own. All ROS packages should reside under the src directory.

Assuming you have already sourced your ROS environment and your catkin workspace (or return to ROS Workspace in the "Introduction to ROS" lesson if you forgot), navigate to the src directory:

$ cd ~/catkin\_ws/src

The syntax for creating a catkin package is simply,

$ catkin\_create\_pkg <your\_package\_name> [dependency1 dependency2 …]

The name of your package is arbitrary but you will run into trouble if you have multiple packages with the same name in your catkin workspace. Try to make it descriptive and unique without being excessively long. Let’s name ours “first\_package” and we won’t specify any dependencies. By convention, package names are lowercase.

$ catkin\_create\_pkg first\_package

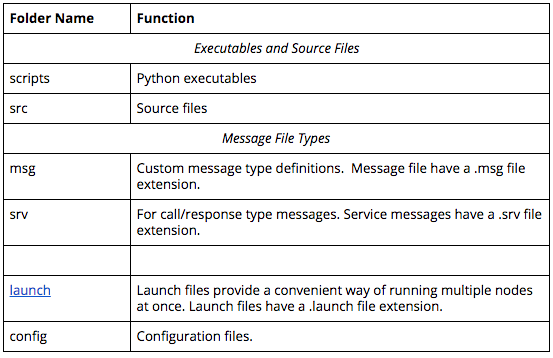
Voilà. You just created your first catkin package! Navigating inside our newly created package reveals that it contains just two files, CMakeLists.txt and package.xml. This is a minimum working catkin package. It is not very interesting because it doesn't do anything, but it meets all the requirements for a catkin package. One of the main functions of these two files is to describe dependencies and how catkin should interact with them. We won’t pay much attention to them right now but in future lessons you will see how to modify them.

I mentioned earlier that ROS packages have a conventional directory structure. Let’s take a look at a more typical package.

* scripts (python executables)
* src (C++ source files)
* msg (for custom message definitions)
* srv (for service message definitions)
* include -> headers/libraries that are needed as dependencies
* config -> configuration files
* launch -> provide a more automated way of starting nodes

Other folders may include

* urdf (Universal Robot Description Files)
* meshes (CAD files in .dae (Collada) or .stl (STereoLithography) format)
* worlds (XML like files that are used for Gazebo simulation environments)

[[](https://classroom.udacity.com/nanodegrees/nd013/parts/30260907-68c1-4f24-b793-89c0c2a0ad32/modules/702b3c5a-b896-4cca-8a64-dfe0daf09449/lessons/f6eaccae-02d3-4274-9428-8cbe39690f90/concepts/9341dc90-e7b7-4e29-a3f7-b6e41834e36f)](https://classroom.udacity.com/nanodegrees/nd013/parts/30260907-68c1-4f24-b793-89c0c2a0ad32/modules/702b3c5a-b896-4cca-8a64-dfe0daf09449/lessons/f6eaccae-02d3-4274-9428-8cbe39690f90/concepts/9341dc90-e7b7-4e29-a3f7-b6e41834e36f)

There are many packages that you can install. To see a list of available packages for the Kinetic distribution, take some time to explore [here](http://www.ros.org/browse/list.php).

1. Recap

