1 – Industial to the course

|  |
| --- |
| 學習目標:   * 如何建立工業機器人的 URDF. * 如何建立 給我們的工業機器人建立MoveIt package. * 如何使用Python來做motion planning. |

2 – Creating the URDF

|  |
| --- |
| **Unified Robot Description Format (URDF)** ( **XML 格式)** 是用來表示robot model.  用URDF 可以表示的東西很多，ex: visual part and how it looks; 物理特性(ex: collision data or the inertia data.)  URDF 在ROS系統裡常常用來表示Robot.  在**ROS-Industrial** 中, 他們還常常用來表示industrial robots, 這章中我們會討論一些特別的東西。 |

|  |
| --- |
| Exercise 1.1  a) 建立一package (my\_robot\_description). 並在裡面建立一urdf folder.  b) 在urdf資料夾裡，建立 **my\_robot.urdf**. 並寫入如下的基本結構:  <?xml version="1.0" ?>  <robot name="my\_robot" xmlns:xacro="http://ros.org/wiki/xacro">  </robot>  c) Add a **world** and a **camera** frame as virtual links.  <link name="world"/>  <link name="camera\_frame"/>  d) Add a **table** frame, specifying both the **visual** and the **collision** tags.  Here you can see an example of a very basic box-shaped table.  <link name="table">  <visual>  <geometry>  <box size="1 1 0.05"/>  </geometry>  </visual>  <collision>  <geometry>  <box size="1 1 0.05"/>  </geometry>  </collision>  </link>  e) Add two joints to connect both the **table** and the **camera** links to the **world** link.  <joint name="world\_to\_table" type="fixed">  <parent link="world"/>  <child link="table"/>  <origin xyz="0 0 0.5" rpy="0 0 0"/>  </joint>  <joint name="world\_to\_camera" type="fixed">  <parent link="world"/>  <child link="camera\_frame"/>  <origin xyz="0 0 1.0" rpy="0 1.571 0"/>  </joint>  註:  In the exmple URDF file, we are adding a camera\_frame because it is a common practice to have a camera in industrial environments such as this.  整個文檔如下:  <?**xml** version="1.0" ?>  <robot name="myrobot" xmlns:xacro="http://ros.org/wiki/xacro">  <link name="world"/>  <link name="table">  <visual>  <geometry>  <box size="1 1 0.05"/>  </geometry>  </visual>  <collision>  <geometry>  <box size="1 1 0.05"/>  </geometry>  </collision>  </link>  <link name="camera\_frame"/>  <joint name="world\_to\_table" type="fixed">  <parent link="world"/>  <child link="table"/>  <origin xyz="0 0 0.5" rpy="0 0 0"/>  </joint>  <joint name="world\_to\_camera" type="fixed">  <parent link="world"/>  <child link="camera\_frame"/>  <origin xyz="0 0 1.0" rpy="0 1.571 0"/>  </joint>  </robot>  截至目前為止，我們建立了一個table在模擬環境裡面。但，robot呢?  一般而言，我們可以將robot也定義在相同的urdf檔裡面，只要加入相對應的links 和joints. (就跟table的case一樣，只是複雜一點點).  然而，在ROS-Industrial中, 建議使用 xacro macros. 所以現在問題就是，what is Xacro?  **Building the Xacro**  Xacro is a macro language (可以讓你很容易的load URDF).  我們前面使用的URDF file 非常簡單，但實際上環境是很複雜的，包含robot，還有一些object model等等。定義這些link和joint會讓你瘋掉@@  使用xacro的好處就是, 我們可以call a macro (如此會自動導入先前已經定義好的URDF檔 ，並讓你的檔案乾淨易讀，而且可以重複使用。  所以，現在就來把robot加入先前建立的環境吧(by a xacro macro) |

|  |
| --- |
| Exercise 1.2  a) 把**my\_robot.urdf file**重新命名為**my\_robot.xacro**.  b) 把含有Motoman robot 的 xacro macro 含入檔案裡面。  該檔案可以在sia10f\_description package 裡頭找到.  <xacro:include filename="$(find **sia10f\_description**)/urdf/sia10f\_macro.xacro" />  c) 所以現在有macro file 去寫入機器人的定義. 但我們還沒有正式建立robot.  所以現在就是去呼叫macro file:  <xacro:motoman\_sia10f prefix=""/>  d) 讓robot跟先前建立的table產生一個 fixed joint  <joint name="table\_to\_robot" type="fixed">  <parent link="table"/>  <child link="base\_link"/>  <origin xyz="0 0 0" rpy="0 0 0"/>  </joint>  e) 建立一個**launch file**. 並在裡面建立 **myrobot.launch**. 在裡面寫入如下內容:  <launch>  <arg name="gui" default="true"/>  <param name="robot\_description" command="$(find xacro)/xacro --inorder '$(find my\_robot\_description)/urdf/my\_robot.xacro'" />  <node name="rviz" pkg="rviz" type="rviz" if="$(arg gui)"/>  </launch>  Here, you are loading your xacro file to the robot\_description parameter, and launching Rviz.  f) 執行 launch file  $ roslaunch my\_robot\_description myrobot.launch  g) Now, just click on the "Add" button, select the RobotModel option, and select "base\_link" as your Fixed Frame. You will now be able to visualize your URDF file!  完整檔案如下:  <?**xml** version="1.0" ?>  <robot name="myrobot" xmlns:xacro="http://ros.org/wiki/xacro">    <xacro:include filename="$(find sia10f\_description)/urdf/sia10f\_macro.xacro" />  <xacro:motoman\_sia10f prefix=""/>  <link name="world"/>  <link name="table">  <visual>  <geometry>  <box size="0.6 0.6 0.05"/>  </geometry>  </visual>  <collision>  <geometry>  <box size="0.6 0.6 0.05"/>  </geometry>  </collision>  </link>  <link name="camera\_frame"/>  <joint name="world\_to\_table" type="fixed">  <parent link="world"/>  <child link="table"/>  <origin xyz="0 0 0.5" rpy="0 0 0"/>  </joint>  <joint name="world\_to\_camera" type="fixed">  <parent link="world"/>  <child link="camera\_frame"/>  <origin xyz="0 0 1.0" rpy="0 1.571 0"/>  </joint>    <joint name="table\_to\_robot" type="fixed">  <parent link="table"/>  <child link="base\_link"/>  <origin xyz="0 0 0" rpy="0 0 0"/>  </joint>  </robot>  一切完成之後，執行以下指令去 reset the robot\_description parameter.  **$roslaunch urdf\_tutorial reset\_robot\_description.launch**  如你所見，Xacro macros 可以輕鬆地讓robot加入URDF file.  你也可以透過URDF輕鬆替換Robot零件，比如加入gripper 到end effector.  只要找到一個xacro macro 去定義gripper，就可以用上述的方法加入.  這樣就可以輕鬆替換不同的end-effector (grippers, articulated hands...).  甚至只用幾行就可以換成新的機器人!  萬事俱備，下一步就來建立MoveIt package! |

3 – Building a MoveIt Package

|  |
| --- |
| **學習目標:**  這一章會學習如何為我們的industrial robot 建立Moveit Package.  並且可以執行motion planning. |

|  |
| --- |
| **Building a MoveIt package**  MoveIt 是一個 ROS framework ，可以讓我們做robot的motion planning .  意思就是讓你可以plan a movement(motion) from a point A to a point B, without colliding with anything.  MoveIt is a very complex and useful tool. 這章不會談太多細節。  (若有興趣可以參考<http://moveit.ros.org/> )  MoveIt 提供了易用的GUI, 可以讓我們與robot互動並執行motion planning.  在使用MoveIt前, 我們會先建立package.  該package會產生所需的configuration 和launch files 來執行我們的robot。 |

|  |
| --- |
| Exercise 2.1  a) 首先, 需要先執行 **MoveIt Setup Assistant**.  執行以下command:   $ **roslaunch moveit\_setup\_assistant setup\_assistant.launch**  結果:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/setup_assistant.png  下一步就是load your robot file  b) Click on the "**Create New MoveIt Configuration Package**" button.  結果:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/load_xacro.png  Now, just click the "**Browse**" button, select the xacro file you created in the Previous Chapter, and click on the "**Load Files**" button. You should now see this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/xacro_loaded.png  所以現在我們已經load robot的xacro file 到 MoveIt Setup Assistant.  c) Go to the "**Self-Collisions**" tab, and click on the "**Regenerate Default Collision Matrix**" button.  You will end with something like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/collision_matrix.png  Here, you are just defining some pairs of links that don't need to be considered when performing collision checking. For instance, because they are adjacent links, so they will always be in collision.  d) Next, move to the "**Virtual Joints**" tab. Here, you will define a virtual joint for the base of the robot. Click the "**Add Virtual Joint**" button, and set the name of this joint to **FixedBase**, and the parent to world. Just like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/virtual_joint.png  Finally, click the "**Save**" button.  Basically, what you are doing here is to create an "imaginary" joint that will connect the base of your robot with the simulated world.  e) Now, open the "**Planning Groups**" tab and click the "**Add Group**" button. Now, you will create a new group called ***manipulator***, which uses the ***KDLKinematicsPlugin***. Just like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/plan_group1.png  Next, you will click on the "Add Kin Chain" button, and you will select the ***base\_link*** as Base Link, and the ***link\_tool0*** as Tip Link. Just like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/plan_group2.png  Finally, click the "**Save**" button and you will end up with something like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/plan_group3.png  So now, you've defined a group of links for performing Motion Planning with, and you've defined the plugin you want to use to calculate those Plans.  f) Now, you are going to create a couple of predefined poses for your robot.  Go to the "**Robot Poses**" tab and click on the "**Add Pose**" button. Your robot will appear with all its joints to "0". So, you will name this pose as **allZeros**, and click the "**Save**" button.  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/allzeros.png  Now, repeat the operation, but this time adjusting the position of the joints so that the robot is in an specific position that we will call "home".  可以建立你想要的動作，但當然是越簡單越好。For instance, something like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/home.png  This is very useful, for instance, when you know that there are some poses that your robot will have to go many times  g) Now, just enter your name and e-mail at the "**Author Information**" tab.  i) Finally, go to the "**Configuration Files**" tab and click the "**Browse**" button.  Navigate to the ***catkin\_ws/src*** directory, create a new directory, and name it ***myrobot\_moveit\_config***. "Choose" the directory you've just created.  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/conf_files1.png  Now, click the "**Generate Package**" button. If everyting goes well, you should see something like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/conf_files2.png  And that's it! You have just created a MoveIt package for you industrial robot.  **Basic Motion Planning**  Well... to start, you can just launch the MoveIt Rviz environment and begin to do some tests about Motion Planning. So, follow the next exercise in order to do so! |

|  |
| --- |
| Exercise 2.2  a) Execute the following command in order to start the MoveIt RViz demo environment.  $ **roslaunch myrobot\_moveit\_config demo.launch**  **NOTE:** It may happen that the Moveit Rviz window appears out of focus. Like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/focus_bad.png  If this is your case, just type the following command into the Web Shell:  wmctrl -r :ACTIVE: -e 0,65,24,1500,550  Now, your MoveIt Rviz window should appear like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/focus_good.png  Now, you can just double-click on the top coloured part of the window in order to maximize.  If everything goes OK, you will see something like this:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/moveit_rviz.png  b) Now, move to the **Planning** tab. Here:  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/moveit_rviz_planning.png  c) Before start Planning anything, it is always a good practice to **update the current Start State**.  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/start_position.png  d) At the query section, you can choose the ***random valid*** option and click on the "Update" button. Your robot scene will be updated with the new random position that has been selected.  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/scene_updated.png  e) Now, you can click on the "**Plan**" button at the "**Commands**" section. The robot will begin to plan a trajectory to reach that random point.  https://i-0935efeb8f2a4645a.robotigniteacademy.com/jupyter/notebooks/img/giphy2.gif  f) Finally, if you click on the "**Execute**" button, the robot will execute that trajectory.  g) Now just play with the new tool! You can repeat this same process some more times.  For instance, instead of moving the robot to a random position, you could set one of the predefined positions you created while building the package (home, allZeros, etc...). You can also try to check and uncheck the different visualization options that appear in the upper "Displays" section.  **Moving the robot in the simulation**  Until now, though, you've only moved the robot in the Moveit application. This is very useful because you can do many tests without worrying about any damage. Anyways, the final goal will be always to move the real robot, right?  我們建立的 MoveIt package 可以提供必要的 ROS **services** 和 **actions** 去規劃與執行軌跡,  但無法將規劃好的軌跡直接丟給實體機器人.  All the Kinematics you've been performing were executed in an internal simulator that MoveIt provides. In order to communicate with the real robot, it will be necessary to do a couple of modifications to the MoveIt package you've created at the beginning of the Chapter.  Obviously, in this Course you don't have a real robot to do this, so you will apply the same but for moving the simulated robot.  In order to see what you need to change in your MoveIt package, just follow the next Exercise. |

|  |
| --- |
| Exercise 2.3 修改Moveit Package，讓規劃好的軌跡可以傳到實體robot  a) First of all, you'll need to create a file to define how you will control the joints of your "real" robot.  Inside the ***config*** folder of your moveit package, create a new file named ***controllers.yaml***. Copy the following content inside it:  controller\_list:  - name: sia10f/joint\_trajectory\_controller    action\_ns: "follow\_joint\_trajectory"    type: FollowJointTrajectory    joints: [joint\_s, joint\_l, joint\_e, joint\_u, joint\_r, joint\_b, joint\_t]  So basically, here you are defining the **Action Server** (and the type of message that it will use) that you will use for controlling the joints of your robot.  First, you are setting the name of your joint trajectory controller Action Server. And how do you know that? Well, if you do a *rostopic list* in any Web Shell, you'll find between your topics the following structure:  https://i-0659f823c28e907f7.robotigniteacademy.com/jupyter/notebooks/img/actionserver.png  So this way, you can know that your robot has a joint trajectory controller Action Server that is called ***/sia10f/joint\_trajectory\_controller/follow\_joint\_trajectory/***.  Also, you can find out by checking the message that this Action uses,  that it is of the type ***FollowJointTrajectory***.  Finally, you already know the names of the joints that your robot uses. You saw them while you were creating the MoveIt package, and you can also find them in the ***sia10f\_macro.xacro*** file, at the ***sia10f\_description*** package.  b) Next, you'll have to create a file to define the names of the joints of your robot. Again inside the ***config*** directory, create a new file called ***joint\_names.yaml***, and copy the following content in it:  **controller\_joint\_names: [joint\_s, joint\_l, joint\_e, joint\_u, joint\_r, joint\_b, joint\_t]**  c) Now, if you open the ***myrobot\_moveit\_controller\_manager.launch.xml***, which is inside the ***launch*** directory, you'll see that it's empty. Put the next content inside it:  **<launch>**  **<rosparam file="$(find myrobot\_moveit\_config)/config/controllers.yaml"/>**  **<param name="use\_controller\_manager" value="false"/>**  **<param name="trajectory\_execution/execution\_duration\_monitoring" value="false"/>**  **<param name="moveit\_controller\_manager" value="moveit\_simple\_controller\_manager/MoveItSimpleControllerManager"/>**  **</launch>**  我們在這就是load 剛剛建立的*controllers.yaml*, 還有*MoveItSimpleControllerManager* plugin,  這可以讓你send計算好的軌跡到 "real" robot.  d) Finally, you will have to create a new launch file that sets up all the system to control your robot.  So, inside the ***launch*** directory, create a new launch file called ***myrobot\_planning\_exeution.launch***.  **<launch>**  **​**  **<rosparam command="load" file="$(find myrobot\_moveit\_config)/config/joint\_names.yaml"/>**  **​**  **<include file="$(find myrobot\_moveit\_config)/launch/planning\_context.launch" >**  **<arg name="load\_robot\_description" value="true" />**  **</include>**  **​**  **<node name="joint\_state\_publisher" pkg="joint\_state\_publisher" type="joint\_state\_publisher">**  **<param name="/use\_gui" value="false"/>**  **<rosparam param="/source\_list">[/sia10f/joint\_states]</rosparam>**  **</node>**  **​**  **<include file="$(find myrobot\_moveit\_config)/launch/move\_group.launch">**  **<arg name="publish\_monitored\_planning\_scene" value="true" />**  **</include>**  **​**  **<include file="$(find myrobot\_moveit\_config)/launch/moveit\_rviz.launch">**  **<arg name="config" value="true"/>**  **</include>**  **​**  **</launch>**  So finally here, we are loading the ***joint\_names.yaml***file, and launching some launch files we need in order to set up the MoveIt environment. You can check what those launch file do, if you want. But let's focus a moment on the *joint\_state\_publisher* node that is being launched.  If you do again a *rostopic list*, you will see that there is a topic called ***/sia10f/joint\_states***.  Into this topic is where the state of the joints of the simulated robot are published.  So, we need to put this topic into the ***/source\_list*** parameter, so MoveIt can know where the robot is at each moment.  f) Finally, you just have to launch this file and Plan a trajectory, just as you learnt to do in the previous exercise. Once the trajectory is planned, you can press the "Execute" button in order to execute the trajectory in the simulated robot.  https://i-0659f823c28e907f7.robotigniteacademy.com/jupyter/notebooks/img/move_sim.gif  下一章就會說明如何用程式來控制motion planning。 |

4 – Motion Planning with Python

這一章裡面會說明如何用程式來控制機器人的移動。

|  |
| --- |
| Demo 3.1  a) 首先先launch the MoveIt Rviz environment. Type the following command:  $ **roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch**  b) Execute the following Python code.  #! /usr/bin/env python  **import** sys  **import** copy  **import** rospy  **import** **moveit\_commander # allow us to communicate with the MoveIt/Rviz interface**  **import** moveit\_msgs.msg  **import** geometry\_msgs.msg  moveit\_commander.roscpp\_initialize(sys.argv) **# initialize the moveit\_commander module**  rospy.init\_node('move\_group\_python\_interface\_tutorial', anonymous=True) **# init node**  robot = moveit\_commander.RobotCommander() **# creating a RobotCommander (Interface of robot)**  scene = moveit\_commander.PlanningSceneInterface() **# interface to the world surrounds robot**  **# interface to the group of joints of robot (i.e: the whole arm)**  group = moveit\_commander.MoveGroupCommander("manipulator")  **# publisher for us to visualize the planned motion through MoveIt/Rviz interface**  display\_trajectory\_publisher = rospy.Publisher('/move\_group/display\_planned\_path', moveit\_msgs.msg.DisplayTrajectory)  **# define a Pose object, that we send as a goal**  pose\_target = geometry\_msgs.msg.Pose()  pose\_target.orientation.w = 1.0  pose\_target.position.x = 0.3  pose\_target.position.y = 0  pose\_target.position.z = 1.1  group.set\_pose\_target(pose\_target)  **# tell "manipulator" group to calculate the plan, if succeed, we can see it on MoveIt/Rviz**  plan1 = group.plan()  rospy.sleep(5)  **# shut down the moveit\_commander module.**  moveit\_commander.roscpp\_shutdown()  c) After a few seconds, you'll see in MoveIt Rviz how the robot is planning the specified motion described in the code above. |

|  |
| --- |
| Exercise 3.1  a) First of all, you'll need to launch the MoveIt Rviz environment. Type the following command:  $ **roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch**  b) Create a **new package** (**my\_motion\_scripts)**.  Inside this package, create a new directory called **src**, with a file named **planning\_script.py**.  Finally, copy inside this file the code you've just executed in the Demo above.  c) Inside the package, create also a **launch** directory that contains a launch file in order to launch the **planning\_script.py** file.  d) Modify the values assigned to the **pose\_target** variable.  Then launch your code and check if the new Pose is achieved successfully.  Repat the process and try with different Poses. |

|  |
| --- |
| **Planning to a joint space goal**  Sometimes, insted of just moving the end-effector towards a goal, we can be interested in setting a goal for an specific joint. Let's see how you can do this. |

|  |
| --- |
| Demo 3.2  a) First of all, you'll need to launch the MoveIt Rviz environment. Type the following command:  $ **roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch**  b) Execute the following Python code:  #! /usr/bin/env python  **import** sys  **import** copy  **import** rospy  **import** moveit\_commander  **import** moveit\_msgs.msg  **import** geometry\_msgs.msg  moveit\_commander.roscpp\_initialize(sys.argv)  rospy.init\_node('move\_group\_python\_interface\_tutorial', anonymous=True)  robot = moveit\_commander.RobotCommander()  scene = moveit\_commander.PlanningSceneInterface()  group = moveit\_commander.MoveGroupCommander("manipulator")  display\_trajectory\_publisher = rospy.Publisher('/move\_group/display\_planned\_path', moveit\_msgs.msg.DisplayTrajectory)  group\_variable\_values = group.get\_current\_joint\_values()  **# Now, we modify the value of 1 of the joints, and set this new joint value as a target.**  group\_variable\_values[3] = 1.5  group.set\_joint\_value\_target(group\_variable\_values)  plan2 = group.plan()  rospy.sleep(5)  moveit\_commander.roscpp\_shutdown()  c) When the code ends executing, you'll see in MoveIt Rviz how the robot is planning the specified motion described in the code above. |

|  |
| --- |
| Exercise 3.2  a) Inside the **my\_motion\_scripts** package, create a new file named **joint\_planning.py**. Do some tests giving different values to different joints.  **Getting some useful data**  Through code, you can also get some valuable data that you may require for your code.  Let's see some examples.  You can get the **reference frame** for a certain group by executing this line:  **print "Reference frame: %s" % group.get\_planning\_frame()**  You can get the end-effector link for a certaing group executing this line:  **print "End effector: %s" % group.get\_end\_effector\_link()**  You can get a list with all the groups of the robot like this:  **print "Robot Groups:"**  **print robot.get\_group\_names()**  You can get the current values of the joints like this:  **print "Current Joint Values:"**  **print group.get\_current\_joint\_values()**  You can also get the current Pose of the end-effector of the robot like this:  **print "Current Pose:"**  **print group.get\_current\_pose()**  Finally, you can check the general status of the robot like this:  **print "Robot State:"**  **print robot.get\_current\_state()** |

|  |
| --- |
| Exercise 3.3  a) First of all, you'll need to launch the MoveIt Rviz environment. Type the following command:  **$ roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch**  b) Create a new file inside the package called **get\_data.py**.  Add all the new code you've learnt just above and check what results you get.  https://i-07e6d4bd33bc031fe.robotigniteacademy.com/jupyter/notebooks/img/get_data_result.png |

|  |
| --- |
| **Executing a trajectory**  So at this point, you've seen some methods that allow you to plan a trajectory with Python code. But...what about executing this trajectory in the "real" robot?  In fact, it's very simple. In order to execute a trajectory, you just need to call the go() function from the planning group. Like this:  **group.go(wait=True)**  By executing this line of code, you will be telling your robot to execute the last trajectory that has been set for the Planning Group. Let's try this in an Exercise! |

|  |
| --- |
| Exercise 3.4  a) First of all, you'll need to launch the MoveIt Rviz environment. Type the following command:  **$ roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch**  b) Create a new Python script called **execute\_trajectory.py**, and copy inside there the code shown in Demo 3.1.  c) Into your new script, add a line in order to execute that trajectory.  d) You can try this with any of the codes you have created for Planning trajectories |

|  |
| --- |
| Exercise 3.5  a) First of all, you'll need to launch the MoveIt Rviz environment. Type the following command:  $ roslaunch myrobot\_moveit\_config myrobot\_planning\_execution.launch  b) Modify the **execute\_trajectory.py** script, so that now it concatenates at least 2 motions. |

5 – Final Project

|  |
| --- |
| 學習目標:  This last Unit is ment to put all the knowledge you've got during this Course together. You will do this by doing a Project, which will be based on the UR5 simulation. Using this simulation, you will have to practice everything you've learned during the Course:   * create an URDF, * build a MoveIt Package, * connect this package to the running simulation * perform some Motion Planning using Python. |

|  |
| --- |
| 1. Build the URDF  First of all, you will have to build an URDF for the robot.  You can follow the steps described in Chapter 1 for that, or you can create a more complex environment, it's up to you! You will find the files you need in the ur\_description package.  Keep in mind, though, that the simulation will not change even if you modify the URDF, because the URDF file of this simulation is already created and launched. Here you have an example URDF file for the project.  https://i-07e6d4bd33bc031fe.robotigniteacademy.com/jupyter/notebooks/img/ur5_urdf.png  2. Build the MoveIt package  Once you have created the URDF file, the next step will be to create a MoveIt package.  Just follow the steps described in the Chapter 2 of this Course.  You should create 3 predefined Poses for the robot. For instance, they could be this 3:    Once the package is created, test that it works properly by launching the demo.launch file, and Planning some trajectories.  It may happen that the trajectories that are shown in MoveIt are a little bit weird. In order to solve this, set the "Planning Attempts" Option to 4, just like in the picture below.  https://i-07e6d4bd33bc031fe.robotigniteacademy.com/jupyter/notebooks/img/attempts.png  At the end, you should be able to see something like this:  https://i-07e6d4bd33bc031fe.robotigniteacademy.com/jupyter/notebooks/img/planning_ur.gif  3. Connect the MoveIt package with the simulation  Once you can successfully perform Motion Planning in MoveIt, it's time to connect it to the simulation in order to be able to move the "real" robot. You can follow the last part of the 2nd Chapter to complete this.  At the end, you should be able to move the robot in the simulation, like this:  https://i-07e6d4bd33bc031fe.robotigniteacademy.com/jupyter/notebooks/img/ur5_sim.gif  4. Python Script  Once everything is working, and you are able to move the robot in the simulation through MoveIt, it's time to do it with code! In this last step of the Project, you will have to create a Python Script that does the following:   * It moves the robot first to one of the predefined Poses you set in Step 2. * Then, it moves the robot to the 2nd predefined Pose. * After this, it goes back to the previous Pose. * Finally, it returns to it's initial Pose. |