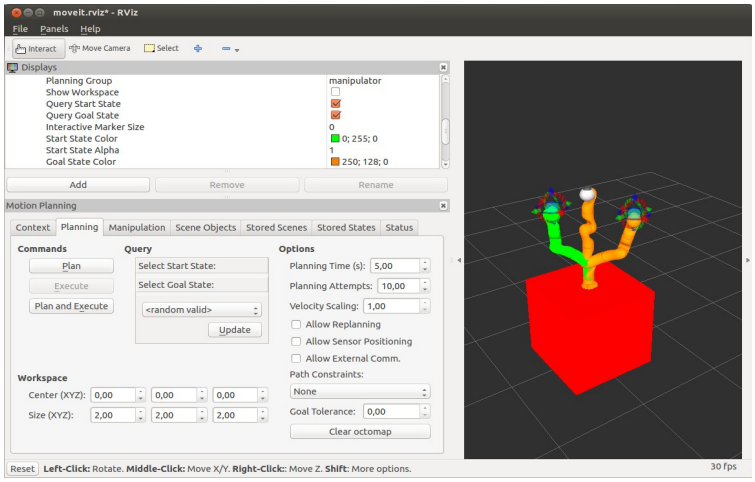
Laurent LEQUIEVRE – Institut Pascal UMR6602

Notions de Move it!, move_group

'MoveIt!' est un ensemble d'outils permettant de faire la planification de mouvement ('motion planning'), de simuler une saisie ('grasping'), de prendre un objet et de le placer ('pick and place'), de faire de la détection de collisions ('collision checking'), de traiter des données de capteurs 3D ('perception'), de la navigation ...

Dans cette partie, on va s'intéresser plus particulièrement à la planification de mouvement.

Un 'plugin' nommé 'MoveIt! Rviz motion planning' a été ajouté dans l'outil de visualisation 'Rviz'. Celui-ci permet de définir graphiquement une position de départ et d'arrivée d'un robot, de lancer le calcul de la planification, d'afficher le résultat. On peut aussi le faire par programmation en utilisant la librairie C++ nommée 'move_group'.



http://moveit.ros.org/documentation/concepts/

Par défaut 'MoveIt' utilise :
- L'outil de planification OMPL ('Open Motion Planning Library' http://ompl.kavrakilab.org/).
- L'outil de détection de collisions FCL ('Flexible Collision Library' https://github.com/flexible-collision-library/fcl)
- L'outil de résolution de la cinématique inverse : The default IK solver ('Inverse Kinematic Solver') in MoveIt! Is a numerical jacobian-based solver.

Ces outils sont vus comme des 'plugin' qui peuvent être utilisés par défaut, ou bien être développés et interfacés par vos soins (par exemple on peut développer son propre outil de planification).

1- Utiliser le plugin 'MoveIt! Rviz motion planning'.

Notre objectif est de créer un paquet 'MoveIt!' qui nous permettra de tester une planification de trajectoire du bras 'kuka' entre 2 poses. Cette planification fera en sorte d'éviter les collisions avec l'environnement, les auto collisions, tout en tenant compte des limites articulaires définies dans le modèle du robot.
Pour se faire, on va utiliser un assistant graphique, pour générer les fichiers paramètres nécessaires

à 'MoveIt!'.

http://docs.ros.org/indigo/api/moveit_setup_assistant/html/_sources/doc/tutorial.txt

Lancer l'assistant :
```
roslaunch moveit_setup_assistant setup_assistant.launch
```

Suivre ces options :
Cliquer sur le bouton '**Create new moveit configuration package**'
Cliquer sur le bouton '**Browse**' et sélectionner le dossier '**model**' du paquet '**kuka_lwr_description**'. Dans ce dossier sélectionner le fichier '**platform.urdf.xacro**' qui contient la description 'urdf' du robot.
Cliquer sur le bouton '**Load files**'
- A ce stade, vous devez voir apparaître le robot sur la fenêtre de droite.

* Onglet '**Self-Collisions**' cliquer sur le bouton '**Regenerate Default Collision Matrix**'
(Permet de calculer/vérifier les collisions entre 'links')

* Onglet '**Virtual Joints**' cliquer sur le bouton '**Add Virtual Joint**' :
- name = '**FixedBase**'
- child link = '**box**'
- Parent Frame name = '**world**'
- Joint Type = **fixed**

Cliquer sur le bouton '**Save**'
(Permet d'attacher le robot au monde)

* Onglet '**Planning Groups**' cliquer bouton '**Add group**' :
- name = '**manipulator**'
- Kinematic solver = **kdl_kinematics_plugin/KDLKinematicsPlugin**
- cliquer sur bouton '**Add Kin. Chain**'
- cliquer sur '**box**' puis sur bouton '**choose selected**' de '**base link**'
- cliquer sur '**kuka_lwr_7_link**' puis sur bouton '**choose selected**' de '**tip link**'
- cliquer sur bouton '**Save**'

(Permet de créer un groupe contenant une partie du robot et de lui associer un 'plugin' solveur cinématique basé sur la librairie KDL http://www.orocos.org/kdl)

* Onglet '**Robot Poses**' – Rien facultatif pour cet exemple
(Permet de définir des poses spécifiques du robot, qui pourront être utilisées avec le 'plugin rviz')

* Onglet '**End effectors**' – Rien facultatif pour cet exemple
(Permet de définir un effecteur au bout du bras, dans notre cas il n'y en a pas)

* Onglet '**Passive Joints**' – Rien facultatif pour cet exemple
(Permet de définir quels 'joints' ne bougent pas, dans notre cas il n'y en a pas)

* Onglet '**Configuration files**' cliquer sur bouton '**browse**', sélectionner le dossier '**src**' du 'workspace' et créer un nouveau dossier nommé '**kuka_lwr_moveit**'.
- cliquer sur bouton '**Generate package**'.
- cliquer sur le bouton '**Exit Setup Assistant**'

(Permet de créer tous les fichiers de configuration nécessaires à 'MoveIt!' dans le dossier

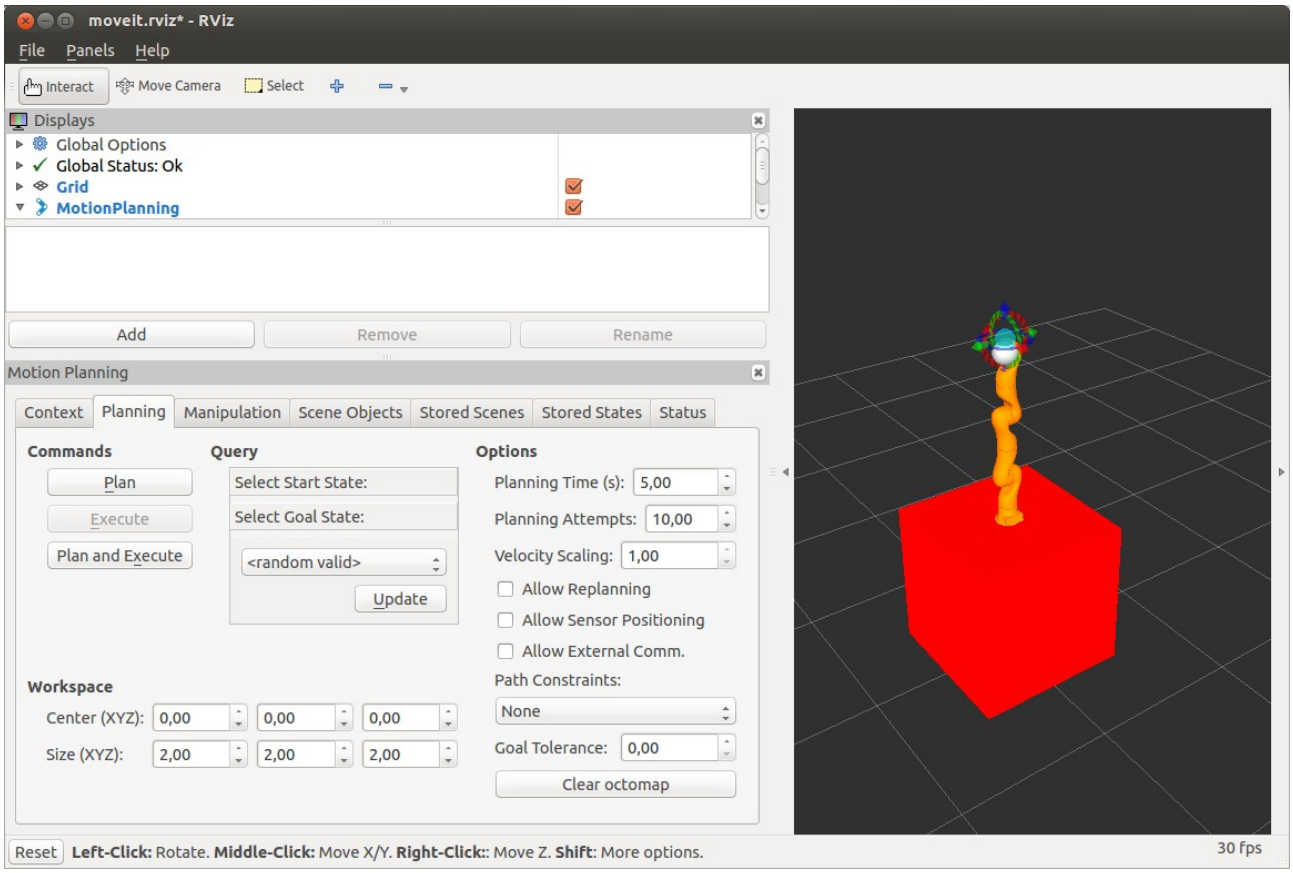
sélectionné)

Relancer une compilation 'catkin' de votre 'workspace'.

Lancer le 'launch' du paquet généré :
roslaunch **kuka_lwr_moveit** demo.launch

Si vous souhaitez modifier la configuration générée dans le paquet 'kuka_lwr_moveit', il faut relancer l'assistant, cliquer sur le '**Edit Existing Moveit …**', cliquer sur le bouton '**Browse**' et choisir le dossier 'kuka_lwr_moveit' du 'workspace' puis cliquer sur le bouton '**Load Files**'.

'Rviz' se lancera avec le plugin 'Moveit', permettant de faire des simulations de trajectoires.

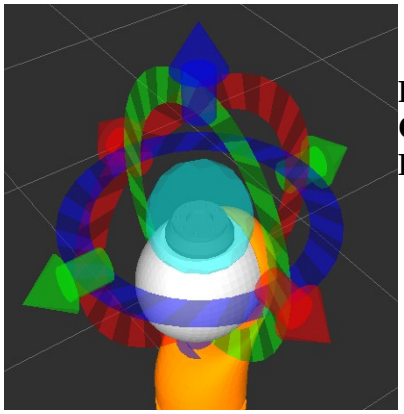


Pour simuler une trajectoire :
Dans la Fenêtre '**Displays**' déployer l'option '**Planning Request**' puis cocher l'option '**Query Start State**' (Observer que le 'start state color' sera par défaut en vert et le 'goal state color' en orange).

* Onglet 'Context' :
- Si la librairie de planification par défaut qui est 'OMPL' (http://ompl.kavrakilab.org/) a bien été chargée, vous devez voir le mot 'OMPL' en vert.

* Onglet '**Planning**' :
→'**Select Start State**' : Se servir des poignées ('markers') pour bouger le bras en vert.
→ '**Select Goal State**' Se servir des poignées ('markers') pour bouger le bras en orange.
(Vous pouvez aussi utiliser l'option 'random valid' puis cliquer sur le bouton 'update')

→ Enfin : '**Plan**'.



Red = Position en X,
Green = Position en Y,
Blue = Position en Z

2- Utiliser l'Api C++ 'move_group' pour interagir avec 'MoveIt!'.

Créer un nouveau paquet nommé 'my_first_move_group' avec des dépendances :

```
catkin_create_pkg my_first_move_group catkin cmake_modules interactive_markers moveit_core moveit_ros_perception moveit_ros_planning_interface pluginlib roscpp std_msgs
```

cf :
http://wiki.ros.org/cmake_modules
http://wiki.ros.org/interactive_markers
http://wiki.ros.org/moveit_core
http://wiki.ros.org/moveit_ros_perception
http://wiki.ros.org/moveit_ros_planning_interface
http://wiki.ros.org/pluginlib

Créer un fichier c++ nommé 'test_random.cpp' permettant la planification aléatoire du bras :

```
#include <moveit/move_group_interface/move_group.h>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "test_random_node", ros::init_options::AnonymousName);
  // start a ROS spinning thread
  ros::AsyncSpinner spinner(1);
  spinner.start();
  // this connecs to a running instance of the move_group node
  move_group_interface::MoveGroup group("manipulator");
  // specify that our target will be a random one
  group.setRandomTarget();
  // plan the motion and then move the group to the sampled target
```

```
 group.move();
 ros::waitForShutdown();
}
```

Ici on crée un 'MoveGroup' qui correspond au '**Planning Groups manipulator**' définit précédemment avec l'assistant :

```
 move_group_interface::MoveGroup group("manipulator");
```

On lui spécifie une position aléatoire à atteindre du bout du bras :

```
group.setRandomTarget();
```

On demande d'effectuer la planification et le mouvement :

```
group.move();
```

Modifier le fichier 'CMakelists.txt' en ajoutant ceci :

```
add_executable(test_random_node src/test_random.cpp)
target_link_libraries(test_random_node ${catkin_LIBRARIES})
```

Lancer dans 2 terminaux différents, 'rviz avec le plugin MoveIt!' et ce nouveau 'node' :

```
roslaunch kuka_lwr_moveit demo.launch
rosrun my_first_move_group test_random_node
```

Observer le mouvement planifié du bras dans 'Rviz'.

Créer un autre fichier C++ nommé 'test_custom.cpp' permettant de spécifier une pose précise à atteindre :

```
#include <moveit/move_group_interface/move_group.h>

int main(int argc, char **argv)
{
 ros::init(argc, argv, "test_custom_node");
 ros::NodeHandle node_handle;
 ros::AsyncSpinner spinner(1);
 spinner.start();

 // The :move_group_interface:`MoveGroup` class can be easily
 // setup using just the name
 // of the group you would like to control and plan for.
 moveit::planning_interface::MoveGroup group("manipulator");

 /*Retrive current position and orientation */
 geometry_msgs::PoseStamped robot_pose;
 robot_pose = group.getCurrentPose();

 geometry_msgs::Pose current_position;
 current_position = robot_pose.pose;

 geometry_msgs::Point exact_pose = current_position.position;
 geometry_msgs::Quaternion exact_orientation = current_position.orientation;

 ROS_INFO("Current position : x=%f, y=%f, z=%f", exact_pose.x, exact_pose.y, exact_pose.z);
 ROS_INFO("Current orientation : x=%f, y=%f, z=%f, w=%f", exact_orientation.x, exact_orientation.y, exact_orientation.z,
exact_orientation.w);

 // Getting Basic Information
 // ^^^^^^^^^^^^^^^^^^^^^^^^
 //
 // We can print the name of the reference frame for this robot.
 ROS_INFO("Reference frame: %s", group.getPlanningFrame().c_str());

 // We can also print the name of the end-effector link for this group.
 ROS_INFO("Reference frame: %s", group.getEndEffectorLink().c_str());
```

```
  // Planning to a Pose goal
  // ^^^^^^^^^^^^^^^^^^^^^^^^
  // We can plan a motion for this group to a desired pose for the
  // end-effector.
  geometry_msgs::Pose target_pose1;
  target_pose1.orientation.w = 1.0;
  target_pose1.orientation.x = 0.0;
  target_pose1.orientation.y = 0.0;
  target_pose1.orientation.z = 0.0;

  target_pose1.position.x = 0.3;
  target_pose1.position.y = 0.3;
  target_pose1.position.z = 1.5;
  group.setPoseTarget(target_pose1);

  // Now, we call the planner to compute the plan
  // and visualize it.
  // Note that we are just planning, not asking move_group
  // to actually move the robot.
  moveit::planning_interface::MoveGroup::Plan my_plan;
  bool success = group.plan(my_plan);

  ROS_INFO("Visualizing plan 1 (pose goal) %s",success?"":"FAILED");

  // then move the group to the sampled target
  group.move();

  ros::waitForShutdown();
  return 0;
}
```

Modifier le fichier 'CMakelists.txt' afin de prendre en compte ce nouveau fichier.
Compiler et tester.

http://docs.ros.org/indigo/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroup-members.html

Tutorial Rviz (http://wiki.ros.org/rviz)