Learning ROS for Robotics Programming

- MoveIt! -

ELISA TOSELLO
toselloe@dei.unipd.it
• About MoveIt!
  • MoveIt! Pipeline
    – move_group
    – Motion planning
    – Kinematics
    – Collision Checking
    – The Planning Scene
  • Integrating an arm in MoveIt!
    – Setup assistant
    – demo.launch – fake controllers
    – Gazebo – controllers
  • Simple Motion Planning tutorials
    – move_group
    – The Planning Scene
    – Kinematics
• **Movelt!** is a set of tools for
  – 3D perception;
  – Motion planning;
  – Kinematics;
  – Mobile manipulation;
  – Trajectory proceeding and execution.

• Install:

  ```bash
  sudo apt-get install ros-kinetic-moveit
  ```
• About MoveIt!
• MoveIt! Pipeline
  – move_group
  – Motion planning
  – Kinematics
  – Collision Checking
  – The Planning Scene
• Integrating an arm in MoveIt!
  – Setup assistant
  – demo.launch – fake controllers
  – Gazebo – controllers
• Simple Motion Planning tutorials
  – move_group
  – The Planning Scene
  – Kinematics
Movelt! PIPELINE

MOVE_GROUP

KINEMATICS

THE PLANNING SCENE

MOTION PLANNING

COLLISION CHECKING
• A **planning group** is a set of joints that need to be planned together in order to achieve a goal;
• The planning is done **separately** for each of the groups.
A motion plan request is sent to the motion planner for a group.

**Goal:**
- Location in the joint space;
- End Effector pose;

**Kinematics Constraints:**
- Position;
- Orientation;
- Visibility;
- User-specified.
The Planning Pipeline searches for a trajectory for all the joints in the group that move the arm so that it reaches the goal.
A trajectory that moves the arm to the target goal location is computed.
The trajectory:

- **Avoids collisions**;
- **Satisfies** the velocity and acceleration **constraints** at the joint level.
• **Forward Kinematics:** (and its Jacobians) is integrated in the **RobotState** class

• **Inverse Kinematics:** MoveIt! provides a default plugin that uses a numerical Jacobian-based solver that is automatically configured by the MoveIt! Setup Assistant.
• Collision checking is configured through the `CollisionWorld` object of the planning scene;

• Collision checking is an expensive operation, then

• **Allowed Collision Matrix:** Matrix used to encode a Boolean value that indicates whether collision checking is needed for two pairs of bodies:
  - value=1: collision checking is not needed (e.g., for bodies that are very far from each other, so they would never collide, or bodies that are adjacent).

(see Setup Assistant)
Components:

- The **world around the robot**
- The **state of the robot**

It is a subpart of **move_group**, which listens to:

- **joint_states**
- Sensor information (e.g., point clouds)
- The world geometry (provided by the user input on the **planning_scene** topic)
• About MoveIt!
• MoveIt! Pipeline
  – move_group
  – Motion planning
  – Kinematics
  – Collision Checking
  – The Planning Scene

• Integrating an arm in MoveIt!
  – Setup assistant
  – demo.launch – fake controllers
  – Gazebo – controllers
• Simple Motion Planning tutorials
  – move_group
  – The Planning Scene
  – Kinematics
For an already integrated robot (e.g., UR5): `roslaunch ur5_moveit_config setup_assistant.launch`
For an already integrated robot (e.g., UR5): `roslaunch ur5_moveit_config setup_assistant.launch`
roslaunch ur5_moveit_config demo.launch
The JointTrajectory controller exposes a controlmsgs::FollowJointTrajectoryAction interface in the follow_joint_trajectory namespace of the controller(*).

(*) http://wiki.ros.org/joint_trajectory_controller
• About MoveIt!
• MoveIt! Pipeline
  – move_group
  – Motion planning
  – Kinematics
  – Collision Checking
  – The Planning Scene
• Integrating an arm in MoveIt!
  – Setup assistant
  – demo.launch – fake controllers
  – Gazebo – controllers
• Simple Motion Planning tutorials
  – move_group
  – The Planning Scene
  – Kinematics
1) Moving to a pose goal
2) Planning to a joint-space goal
3) Planning with Path Constraints
4) Cartesian Paths
5) Adding Objects
6) Planning without Collision
7) Attaching Objects
8) Detaching Objects
9) Removing Objects
BASIC CONCEPTS:

- The planning scene publisher publish over a topic using “diffs”;
- A planning scene **diff** is the difference between the current planning scene (maintained by the move_group node) and the new planning scene desired by the user.

COMMON COMMANDS:

- Adding and removing objects into the world;
- Attaching and detaching objects to the robot.
THE PLANNING SCENE
Simple Motion Planning

IAS-LAB
The `RobotModel` and `RobotState` classes are the core classes that give you access to the kinematics.

- **moveit::core::RobotModel**: definition of a kinematic model;
- **moveit::core::RobotState**: representation of a robot’s state. This includes position, velocity, acceleration and effort.