Robot description (URDF)

This should exist on the parameter server. Most applications that use this description assume the parameter name is 'robot_description'. Since this may not be the name, the description should be remapped.

pr2_defs is the package that contains this description and a launch file that sends the description to the parameter server.

Collision & planning description

In addition to the robot description we also need to know which robot links need to be checked for collision (and with what padding, scaling), which links the robot can see with it own sensors (so we can clear point clouds of points on self). For planning purposes, we need to define groups of links for which we plan for. The parent joints of the links are the the joints we actually control.

This information exists as YAML files in the pr2_defs package. A launch file that sends this information to the parameter server is also available. The description of the content for these YAML files is available in the documentation for the planning_environment package.

Perception

For motion planning to work, sensor data about the environment must be available. Lasers or cameras can be used; what matters is that mapping_msgs::CollisionMap and/or mapping_msgs::ObjectInMap messages get to the planning environment. For the robot state, robot_msgs::MechanismState is needed.

Example launch files that start perception are in the tabletop_scripts package.

Collision Map

Create a collision map from the received point cloud. This code is in the collision_map package.

Robot Self Filter

Add a channel in the pointcloud that marks which points are on the robot and which are actually points on the obstacles. This code is in the robot_self_filter package.

Clear Known Objects

Remove points in the pointcloud that correspond to known objects. The code for this is in the planning_environment package.

Sensor

Produces point cloud information

Planning Models

Planning models are representations of the robot we want to perform motion planning for. These models should be placed in the planning_models package. Currently, only a kinematic model (planning_models::KinematicModel) is available. This model is able to perform forward kinematics for groups of joints (the ones we want to plan for) and compute the transforms for every link of the robot in the frame of the link that attaches the robot to the environment. A parsed robot description needs to be provided as input so that a robot model can be constructed. A state of the kinematic model (set of joint values) can be maintained using the planning_models::StateParams class.

Usually, models in this package should not be used directly, but only through the planning_environment::RobotModels class from the planning_environment package, which provides an instance of each existing robot model, based on data from the parameter server.

Collision Spaces

The collision_space package contains definitions of environment models. These environment models perform collision checking between the robot and the obstacles in the environment and self collision checking. Different back-ends are possible, depending on the collision checker used behind the scenes (ODE, Bullet). The user should however use only the provided abstract interface. An instance of a planning_models::KinematicModel needs to be provided so that the positions of the robot's links can be computed. Environment models can be cloned, which allows performing collision checking in parallel.

Usually, models in this package should not be used directly, but only through the planning_environment::CollisionModels class from the planning_environment package, which provides an instance of every existing collision model, based on data from the parameter server.

Planning Environment

The planning_environment package creates robot models and collision space models for use with motion planning, using information loaded on the ROS parameter server. All information specified by the collision & planning descriptions is available as well. In addition to these models, monitors for various models are available as well. These monitors listen to relevant ROS topics to maintain things such as the current robot state or the current state of the collision environment.

In general, applications that perform motion planning should make use of the available monitors (usually planning_environment::PlanningMonitor). Please read the documentation for the planning_environment package as well.

Moving Action

This is a robot action (such as the move_arm package) that allows the robot to move to a desired goal. Th motion planner is asked for a path, which then gets sent to the controller. As the robot is moving, the environment is monitored and if the path becomes invalid, the controller is asked to stop the path execution and the planner is asked to compute a new path.

When planning for an arm to a goal location in space, inverse kinematics may be used as well.

Example launch files for starting the moving action and corresponding controllers, planning node and inverse kinematics are in the tabletop_scripts package.

Executive

This is where the code that decides which goals we are to pursue lives.

Planning Node

This must be a not that provides a service for motion planning. There are multiple packages that perform this function: ompl*, sbpl*, chmp*

Inverse Kinematics

If desired, inverse kinematics can be used to find joint angles at the goal location.

Trajectory Controller

This is a controller that can execute a trajectory produced by the motion planner. The controller must be able to report which joints it acts upon and must be able to cancel trajectories that are currently being executed.

Object Recognition

This is where code that can perform object recognition lives. For example, the recognition_lambertian package or the table_object_detector package.