Robot Motion Planning

Introduction to MoveIt!

Guillaume Walck

January, 2015

Outline

Outline

Scenario

”Robot, put this apple into the basket”

- Understanding the request
- Understanding the environment
- Planning the task
- Executing the task

Objective

Planning the task

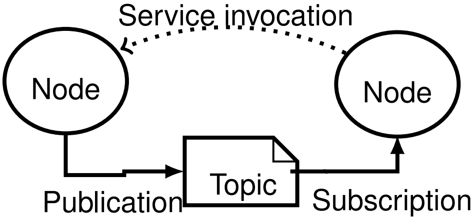
- Task decomposition (skills)
  - Reaching
  - Grasping
  - In-hand manipulation
- Motion planning
  - Motion primitives decomposition
  - Trajectories generation
  - Reactive / servoed motion
  - Constrained movements
  - Obstacle avoidance

Robotic framework : ROS

- What is ROS
  - Hardware abstraction for robots
  - Inter-process communication
  - Package management system
  - Development and deployment ease
- ROS Features
  - Multi-language (C++, Python, Lisp, Java soon)
  - Modular and re-usable
  - Peer-to-peer communication
  - Open-source (BSD license mostly)

ROS Concepts

- Package: contains nodes (code), data, config, etc...
- Master: registers topics and services, enables peer-to-peer com init.
- Nodes: computing components that exchange messages over topics/services
- Messages: description of data that transit between nodes
- Topics: bus with a name to share messages (publish/subscribe)
- Services: description of request/answer exchange between nodes
- Parameter server: central parameter location



MoveIt! overview

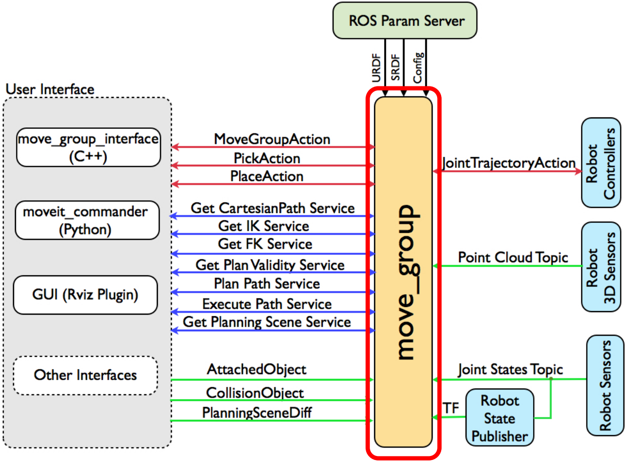
Mobile manipulation software in ROS

- Features
  - Motion planning
    - Navigation
    - Manipulation
  - Environment integration (3rd party lib)
    - 3D perception
    - Kinematics
    - Control
- Many robots supported
  - Mobile platforms
  - Arms with grippers
  - Bimanual setups
  - Humanoids

Outline

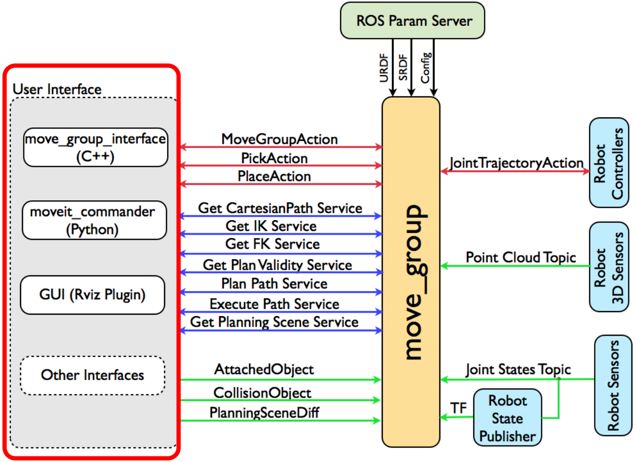
MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)



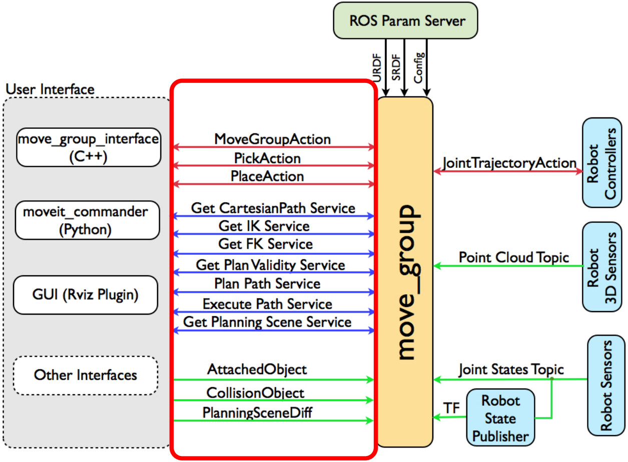
MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)



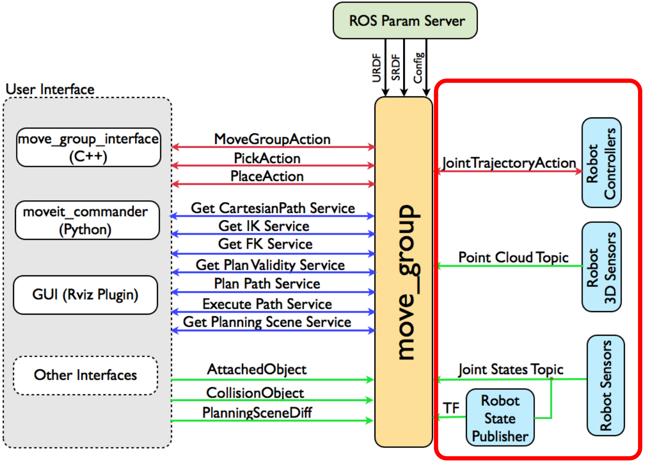
MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)



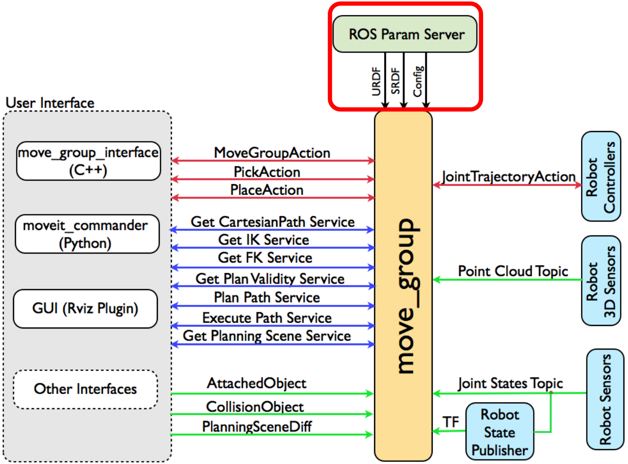
MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)



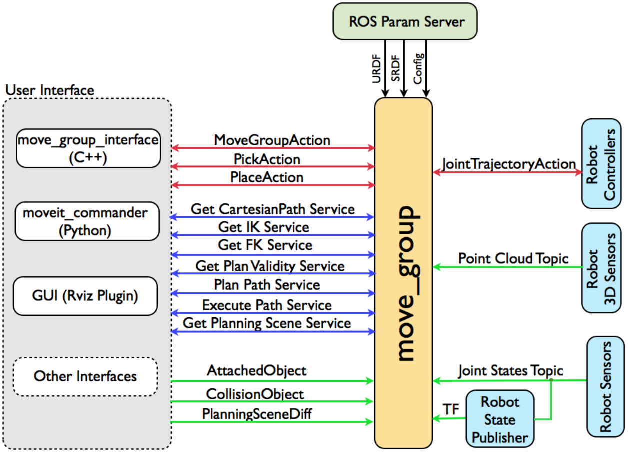
MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)

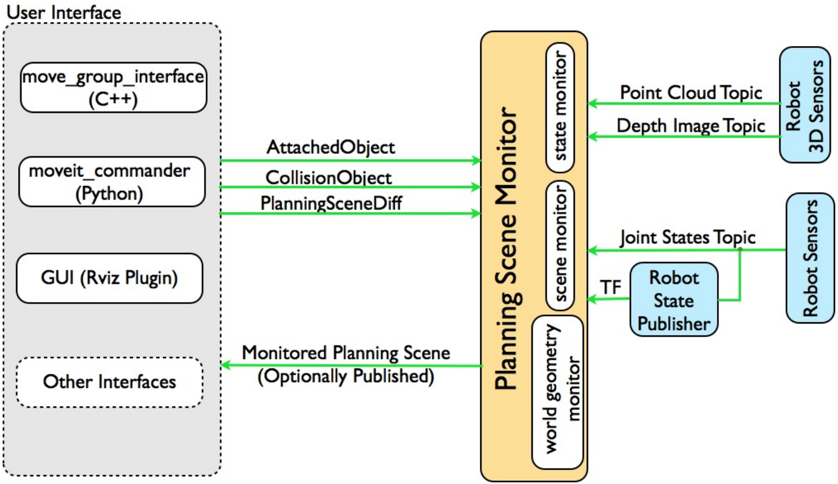


MoveIt! system architecture

- move_group node
- UI (C++ / Python / GUI)
- ROS Interface
- Robot Interface (JS / TF / CTRL ...)
- Config (URDF / SRDF / Wizard)
- Plugins (planning :OMPL collision: FCL kinematics : KDL)



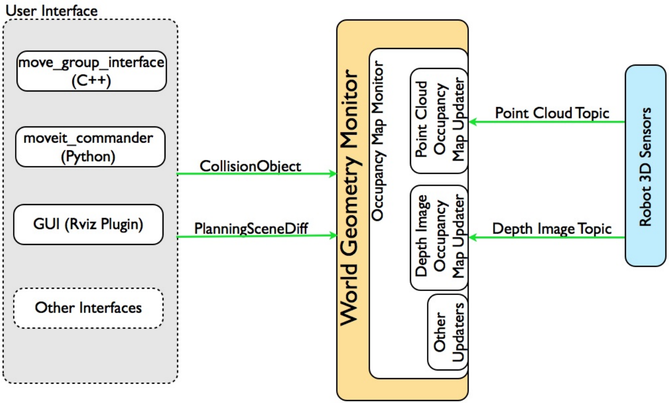
Planning Scene: representation of the environment



3D Perception

World geometry monitor

- Occupancy Map (Point cloud / Depth images)
- Use of octomap
- Self filtering



Motion planning libraries

- OMPL (Open Motion Planning Library) [1]
  - Sampling-based motion planning
  - Probabilistic (PRM), Tree-based (RRT, EST, SBL, KPIECE)
  - Interfaced and configured by MoveIt!
  - Collision checking through 3rd party lib
- CHOMP (Covariant Hamiltonian Optimization and Motion Planning) [2]
- SBPL (Search-based planning) [3]

[1] Ioan A. Şucan and Mark Moll and Lydia E. Kavraki IEEE Robotics & Automation Magazine, 2012
[2] N. Ratliff. and M. Zucker and J.A. Bagnell and S. Srinivasa, ICRA, 2009
[3] Maxim Likhachev, CMU

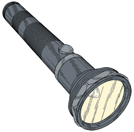
Kinematics & Collisions

- Kinematics
  - Orocos KDL
  - Basic FK + Jacobians
  - IKFast link
  - Own IK
- Collisions
  - Flexible Collision Library
  - Meshes, Primitive Shapes, Octomap
  - Allowed Collision Matrix

What about hand motion planning ?

- In-hand manipulation
  - Button pressing/switching/turning
  - Precise grasping
  - Regrasping
  - Object moving



Outline

Tutorial organization

- Tutorial 1: MoveIt! for dexterous manipulation planning
  - Setting up the Shadow hand config in MoveIt!
  - Topic addressed
    - Using the wizard for a dexterous hand
    - Solving IK for non-6D capable effectors
- Tutorial 2: Using the planning environment
  - Using the setup for in-hand manipulation
  - Topic addressed
    - Multifinger planning with interactive markers

Tutorials setting up

The tutorial is available online in a wiki page. The steps are shown on the slides but detailed on the wiki

The URL is
`http://github.com/ubi-agni/dexterous-manipulation-tutorial/wiki`

Tutorial 1 MoveIt! config with dexterous hands

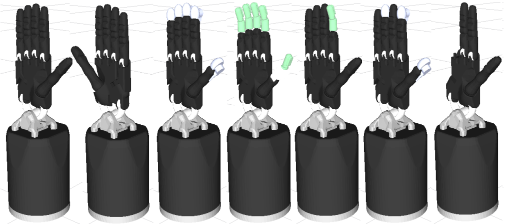
This tutorial shows how to setup a planning environment for a multi-fingered hand or an advanced gripper.

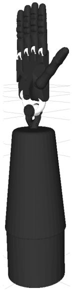
- Fingers are mini-serial manipulator
- MoveIt! can plan trajectories for serial manipulators
- Special settings needed

Understanding the robot description (URDF)

MoveIt! configuration wizard requires a URDF file of the robot to generate a semantic robot description format file used for planning

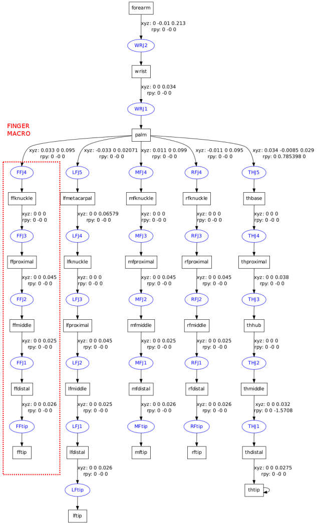
- The Shadow robot hand URDF files for various versions
  - Motor or Muscle driven
  - Equipped with standard, biotac or ellipsoid fingertips
  - With 1 to 5 fingers
  - Left or right handed





Understanding the robot description (URDF)

- Modular description
  - Generated through XACRO (XML macro language)
  - Each element in a different xacro file
  - Parameters to set the various versions
  - Macros called by wrappers in a hierarchical manner
    finger phalanxes compose a finger which compose the hand along with a palm, a wrist and a forearm
- Tip links necessary for IK



Using the wizard for the Shadow hand

7 Steps

1. Loading the URDF file
2. Generating and checking the collision matrix
3. Adding virtual joints
4. Creating planning groups
5. Adding robot poses
6. Adding end-effectors
7. Generating the config files

IK for non-6D capable effectors

Can we use KDL generic IK solvers ?

- Not for coupled joints :
  KDL can solve serial chains but does not handle coupled joints.
  Processed as separate joint $\Rightarrow$ some IK solutions not possible on the robot.

- 3D only solver
  Option to set KDL to solve position-only IK (3D IK) now available.

Tutorial 2 : Using the planning environment

This tutorial shows how to plane multi-fingered trajectories with interactive markers.

Multi-fingered planning with interactive markers

- Planning multi-fingered motion requires multiple goal states to be defined one goal for each fingertip $\Rightarrow$ cumbersome
- One idea would be to virtually link several tips together to change the state as a set of tips.

Interactive markers will be used for this purpose

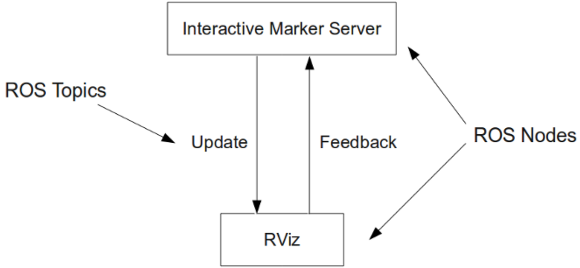
Understanding interactive markers

Interactive markers were already used in RVIZ to move the end-effectors.
What are the interactive markers ?

- Markers displayed in RVIZ, the user can interact with, changing their properties:
  - position
  - orientation
  - menu entries
- Different types exist
  - 6 DOF (fixed or relative)
  - 3 DOF
  - Quadracopter
  - Menu
  - ...see the Basic Control Tutorial for all the types

Understanding interactive markers

How do interactive markers work

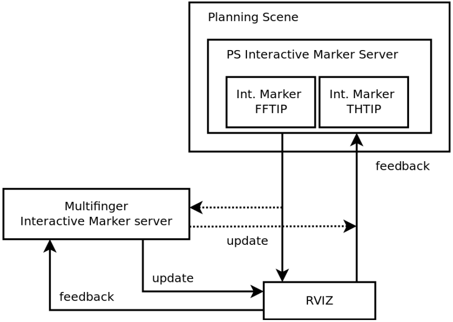


- Structure properties:
  - A feedback topic is published by the display client to the marker server, each message holds the property changes of the marker (event)
  - An update topic to modify the marker status from the server to the display client image Processing the user action is done in a callback on incoming feedback messages

Multifinger planning concept

Interfacing MoveIt with new interactive markers

- Send simulated clicks (feedback) to the tip markers.
- Listen to tip markers update.
  The state (start or goal) currently controlled is stored in the update message as:
  marker_name: EE:goal_fftip



Development

These major steps are described in the tutorial via code snippet comments

- Processing the feedback
- Getting which state is controlled in the planning scene
- Menu feedback processing
- Creation of the marker
- Menu marker
- Main loop

Tests

Test in demo mode of MoveIt!

1. Compile the multifinger_planning_marker
2. Starting MoveIt! demo
3. Run the multifinger_planning_marker
4. Add an interactive marker plugin
5. Select the first_finger_thumb group
6. Activate the multifinger marker
7. Set the goal and start states
8. Plan the motion

Outline

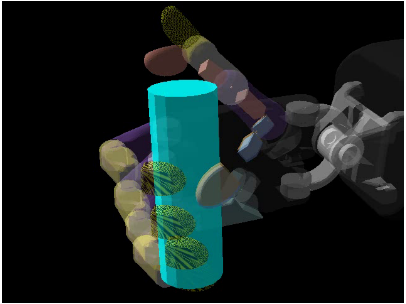
Lessons learnt

- Motion planning is a significant step in task planning
- Motion planning relies on perception (proprio and exterio)
- Motion planning is nicely integrated within MoveIt!
- MoveIt! can plan arm motion and multi-finger motion
- User interface is quite flexible, with interactive markers

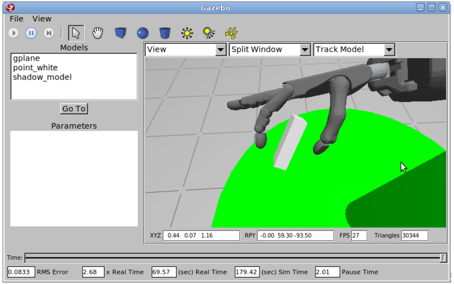
Further possibilities ?

A few examples in dexterous manipulation planning

- Consider obstacles
  An object in-hand can be used as an obstacle to replace fingers without colliding with the object



- Plan for virtual degrees of freedom
  Arm + palm + 6 virtual DOFs
  ⇒ Traj. for virtual DOF as input to in-hand planning (finger gaiting)



Exercise

MoveIt! wizard for an arm and hand configuration with file
/vol/nirobots/ros/indigo/share/agni_description/robots/left_pa10_shadow_gazebo.urdf.xacro

- Add at least 2 end-effectors
  - Arm only
  - Arm and palm
  - Bonus : Arm and palm and first fingertip
- Use KDL solver
- Do not optimize the collision matrix
- Try some planning
- Tip : Generate the urdf file and look at it first to understand the different links and where arm starts, where hand starts etc...