HANDS-ON 6: ADAPTING THE MANIPULATION STACK TO IN-HAND MANIPULATION

Guillaume Walck

guillaume.walck@upmc.fr

IROS 2012

October 7th 2012
Outline

1. Object manipulation stack overview
2. Integration on your robot
3. Towards in-hand manipulation
4. Integrated example
5. Conclusion
Object manipulation stack overview

- Introduction
- Stack Description
- Structure
- Previous session recap
Introduction

Context

- ROS Electric on Ubuntu Lucid, Natty and Debian Squeeze
- Presentation of the stack from User point of view
- Future version called Move It! not treated here
- Based on HANDLE project development experience
- Providing feedback on common or specific issues
Stack Description

What is it?

Current version of packages for object manipulation with any robot manipulators to perform object pickup and placing.

- **Main Stack**
  - `Household_objects_database`: mesh & grasps
  - `Object_manipulator`: core application
  - `Probabilistic_grasp_planner`: grasp generator

- **Major Dependencies**
  - `Arm_navigation`: dealing with arm movements
  - `Motion_planning`: access to planning libraries
  - `Kinematics`: computing robot kinematics
  - `Sql_database`: access to database
Structure

Object manipulation stack overview
Introduction
Stack Description

Object manipulation stack overview

Guillaume Walck
Hands-on 6: Adapting the manipulation stack to in-hand manipulation
Subsets

- Environment sensing (vision in Hands-on 1 & sensing in Hands-on 4)
- Environment analysis (transforms in Hands-on 2)
- **CollisionMap** processing (in Talk 3)
- Arm navigation (in Talk 3)
- Object manipulation (in Talk 3)
- Drivers (in Hands-on 3)
Previous session recap

Hands-on 1 Visual perception system

- Kinect: understanding and ROS integration
- System for robotic visual perception
  - Calibrating a Kinect with respect to the robot
  - Reconstruction
  - Object recognition: known objects stored in a database
  - Pose estimation of the object
- Interfacing with an object SQL database
Hands-on 2 Managing coordinate frames

- Reference Frames: Visualizing the tree
- Creating static transforms
- Transform broadcast → publishing transforms
- Robot urdf model
- Transform listener → LookupTransform
- Transform Point/Vector/Pose
- Transforms and time-keeping → Application in object tracking
Previous session recap

Hands-on 3 Control of the Shadow dexterous hand

- Ethercat motor hand
- 1KHz realtime loop updates:
- Hand HW drivers
- Controllers
- User interaction with controllers using command topic
Integration on your robot

- Pre-requirements
- Wizard
Overview

- **URDF**: handling several robots
- **TF**: linking robots and timing issues
- **Control**: `robotstate` and controllers
- **Sensing**: tactile & vision
- **Database**: objects & grasps
URDF: handling several robots

- **Robot description**
  - Required by many nodes: TF, IK, planning, visualization
  - Keep it modular: robot separated from platform
  - Dealing with 2 linked robots: hand & arm
  - Simplify collision models

- **Simulation world**
  - Platform: model attached to world link
  - Objects: realistic but should remain lightweight models
  - Scenarios: launch files spawning several objects

---

Guillaume Walck

Hands-on 6: Adapting the manipulation stack to in-hand manipulation
Problem: 2 robots creating a complete manipulator

- **Single robot**
  - 1 complete `joint_states` from end-to-end
  - 1 `state-publisher` with a full URDF

- **Multiple robots**
  - 1 `joint_states / state-publisher` per robot
  - 1 `joint_states merger` (full js needed)
  - 1 `static_transform` to link the robots together
  - Full URDF still loaded as `robot_description`

⇒⇒ Coherent TF linking equivalent to full URDF
⇒⇒ Check for no duplicate TF state-publishing

---

Guillaume Walck

Hands-on 6: Adapting the manipulation stack to in-hand manipulation
Pre-requisites

TF: timing issues

- **Context**
  - 4 computers (hand, arm, vision, manip)
  - TF published by 3 of 4 computers (arm, hand, vision)
  - TF used by all the computers

- **Requirements**
  - Coherent URDF / State Publishers (non overlapping)
  - Synchronization of time

⇒ NTP daemon is MANDATORY
Control: robotstate (1)

Robotstate with 2 robots on 2 different buses linked together?

- **YES:** robots based on torque commands
  - Ethercat drivers & non-ethercat drivers in real-time loop
  - One `pr2_controller_manager` handles all the controllers
  - Robotstate containing joints/actuators from end-to-end
  - All controllers in `robot_mechanism_controllers` can work

- **NO:** robots based on mixed type commands
  - Robotstate is only dealing with torque demands
  - No controller in `robot_mechanism_controllers` can work
Pre-requirements

Control: robotstate (2)

- Solution: use velocity/position commands
  - Drivers fill own `robotstate` in separate `realtime loops`
  - Multiple `CM + service (listcontrollers, ...)` dispatching node
  - No generic `robotstate` ➞ create a `getJointState` service
  - Custom designed controllers (ex: muscle pressure commands)
Pre-requirements

Control: controllers

- Types required by `move_arm` execution
  - Joint trajectory action controller
  - Cartesian space controller

- High-level control loop for non-torque controlled multi-robots
  - Manages the `joint_trajectory` or cartesian controllers
  - Connects velocity/position commands output to low-level controllers input topics
  - Running at 100-500Hz

Mount: controllers

- Control: controllers
  - Types required by `move_arm` execution
    - Joint trajectory action controller
    - Cartesian space controller
  - High-level control loop for non-torque controlled multi-robots
    - Manages the `joint_trajectory` or cartesian controllers
    - Connects velocity/position commands output to low-level controllers input topics
    - Running at 100-500Hz
Pre-requirements

Sensing: vision & tactile

- **Vision device**
  - Pointcloud acquisition
  - Calibration mean
  - Self-filtering

- **Tactile device**
  - Force and contact measuring
  - Gripper/hand grasp quality
  - Reactive actions
Pre-requisites

Database: objects & grasps

- Objects
  - Properties
  - Mesh model

- Grasps
  - Related to objects
  - Pre-grasp posture
  - Grasp pose
  - Grasp posture + quality

Object with symmetries with non-360 end-effectors
  - Bad object frame setup after recognition
  - Planning does not consider symmetry
  \[\Rightarrow\] Generate poses around symmetry axis on-the-fly
Wizard

Hints

- **Setting-up**
  - Add all the possible chains even if you don’t use them
  - Adapt URDF files pointers (in case you use on-the-fly URDF generation from xacro)
  - Edit constraint_aware_kinematics.launch to add your own IK services (must be plugins)

- **Testing**
  - Launch the create_launch_files.py from move_arm_warehouse
  - Try the result in the generated warehouse_viewer application
3 Towards in-hand manipulation

- Comparison
- Adapting nodes
- Creating nodes
Comparison

Similarities

- **Pick & Place**
  - Is an in-hand manipulation requirement
  - Approach/lift phases similar

- **Planning**
  - Fingers are mini serial manipulators
  - Object is added as an obstacle for the fingers

- **High-level control**
  - Joint trajectory controller works on fingers
Differences

- **Object on table vs in hand**
  - Tracking in-hand
  - Planning very close to objects

- **Context only vs task & context oriented grasps**
  - Task pre-conditions final grasp
  - Context pre-conditions initial grasp
  - In-hand transition planning

- **Gripper vs fingers grasps**
  - N-fingers
  - Grasp synergies
  - Force closure
  - Contact & slip detection
To cope with the similarities and differences:

- Existing nodes need to be adapted
- New nodes must be created
Adapting nodes

IK

- Manipulator: arm + wrist
  - Multi-robot but single URDF
  - Arm_kinematics_constraint_aware package
  - 6D IK generated automatically by KDL

- Hand: 5 fingers (<6 DoF)
  - Modify generic arm_kinematics node to solve 3D IK
  - Adapt KDL library to support coupling
  - Kinematics_base to derive a plugin usable in OMPL
Adapting nodes

Grasps

- **Access & description**
  - Access node `household_objects_database` totally re-used
  - Grasp format configured through `hand_description` parameter
    ➞ Never use "hand_description" word for an URDF describing a hand!

- **Structure**
  - Double precision vector in radians for joint angles
  - Pose given with translation and quaternions
    ➞ quaternion values are \((qw,qx,qy,qz)\) in DB but \((qx,qy,qz,qw)\) in ROS message

- **Execution**
  - Basic interpolation between pre-grasp posture and grasp-posture
  - Rewriting `gripper_action` into `hand_posture_execution`
Adapting nodes

Finger movement planning

- **Finger gaiting**
  - To reconfigure the grasp
  - To avoid object parts with in-hand movements
    → Re-use **OMPL** node with custom IK

- **Example: Moving the thumb in a power grasp**
  - Plan the thumb movement to avoid object collision
  - Small padding needed
**Creating nodes**

**IK**

- **KDL** creates iterative slow solutions
  
  ➔ Analytic solution is better for faster planning

- **Openrave** IKFast
  
  ▶ Convert robot description from **URDF** to **DAE**
  
  ▶ Generate cpp code
  
  ▶ Integrate the code in a **kinematic_base** inherited plugin
Creating nodes

Reactive Grasping

- Replaced by **synergy grasping**
  - Works on known & unknown objects
  - Stops each finger independently at contact
- Indirectly ensure force-closure
  - Impedance control on each fingertip in contact
  - Self-adjusts to stable grasp

---

Each Finger is stopped independently when contact is detected

Synergies for a tripod grasp

Impedance force control
Creating nodes

In-hand planning

- Core application of the project
  - Learning transitions and grasps from Humans
  - Planning grasp transitions and action steps
  - Benefit from fingers dexterity
  - Synergy and geometric movement planners

- Integration
  - Services mostly triggered by manipulation stack messages
  - Using `actionlib` to easily cancel actions (object slippage)
Creating nodes

Pick, Manipulate & Place

- Initial node
  - **Object_manipulator_node** is very complex
  - Too many cases hardcoded
  - Over protection difficult to remove

- Rewriting
  - Python application
  - Separate planning from execution
  - Redo basic pick and place
  - Add intermediate steps (grasp transitions)
  - Do something useful (action)
4 Integrated example
- Manipulation
- In-hand manipulation
Context

- **HANDLE Platform**
  - Shadow biomorphic arm
  - Shadow EDC motor hand
  - Kinect
  - ATI Nano17 F/T sensors

- **Processing power**
  - Far from PR2 power
  - 2 PC for robot/sensor drivers
  - 1 PC for image processing
  - 1 PC for manipulation stack
- Detect the can
- Extract a grasp, generate a symmetry
- Plan the approach with OMPL
- Perform the reactive grasp (interpolation)
- Lift (Place not working yet)
Object handling (1)

Warehouse_viewer with object attached to the hand

Insert a book in a slot (SIM)  Insert a book in a slot (REAL)
Warehouse_viewer with object attached to the hand

Insert a book in a slot (SIM)  
Insert a book in a slot (REAL)
Warehouse_viewer with object as obstacle

Thumb gaiting
Summary of tips & tricks

- Whenever possible, use existing messages/services

- Describe your robot (URDF, robotstate, controllers, IK)

- Use provided wizards and re-use generic nodes (KDL, OMPL)

- Fill up the database for your application/hand

- Whenever possible store data in existing tables

- Complete the pipeline with missing nodes for your application
Manipulation stack

- **Pros**
  - Complete pipeline
  - Quite configurable (wizards)
  - Various useful standalone external apps
  - Very re-usable

- **Cons**
  - Complex, not documented enough
  - Still PR2 oriented, not generic enough
Our developments always behind due to high pace of the main packages

Complete with a lot of solutions ported to ROS

Great sharing community

⇒ definitely time saver
Questions?

Any questions?