Arm Navigation Tools

From URDF -> Databases and Execution

E. Gil Jones
Willow Garage

What is arm-navigation?

- A set of stable components for:
  - Forward kinematics
  - Robot and environment representation
  - Self and environment collision and constraint checking
  - Inverse kinematics
  - Sampling-based trajectory generation
  - Trajectory smoothing and short-cutting
  - Trajectory execution and monitoring
  - Voxelized and probabilistic world modeling
- And also:
  - Tools for configuring and visualizing the above for your robot
- As well as many experimental features
  - Database-enabled Arm Navigation

What isn't arm-navigation?

- Much help for creating the physical specification for your robot
- Much help for writing controllers for your robot
- A instantaneous generator of optimal dynamic trajectories for your robot
- A guarantee that your robot won't ever hit anything
- A solution to the perception problem
- A closed-source, finished product
    - We welcome contributions of all kinds!

Rest of this talk:

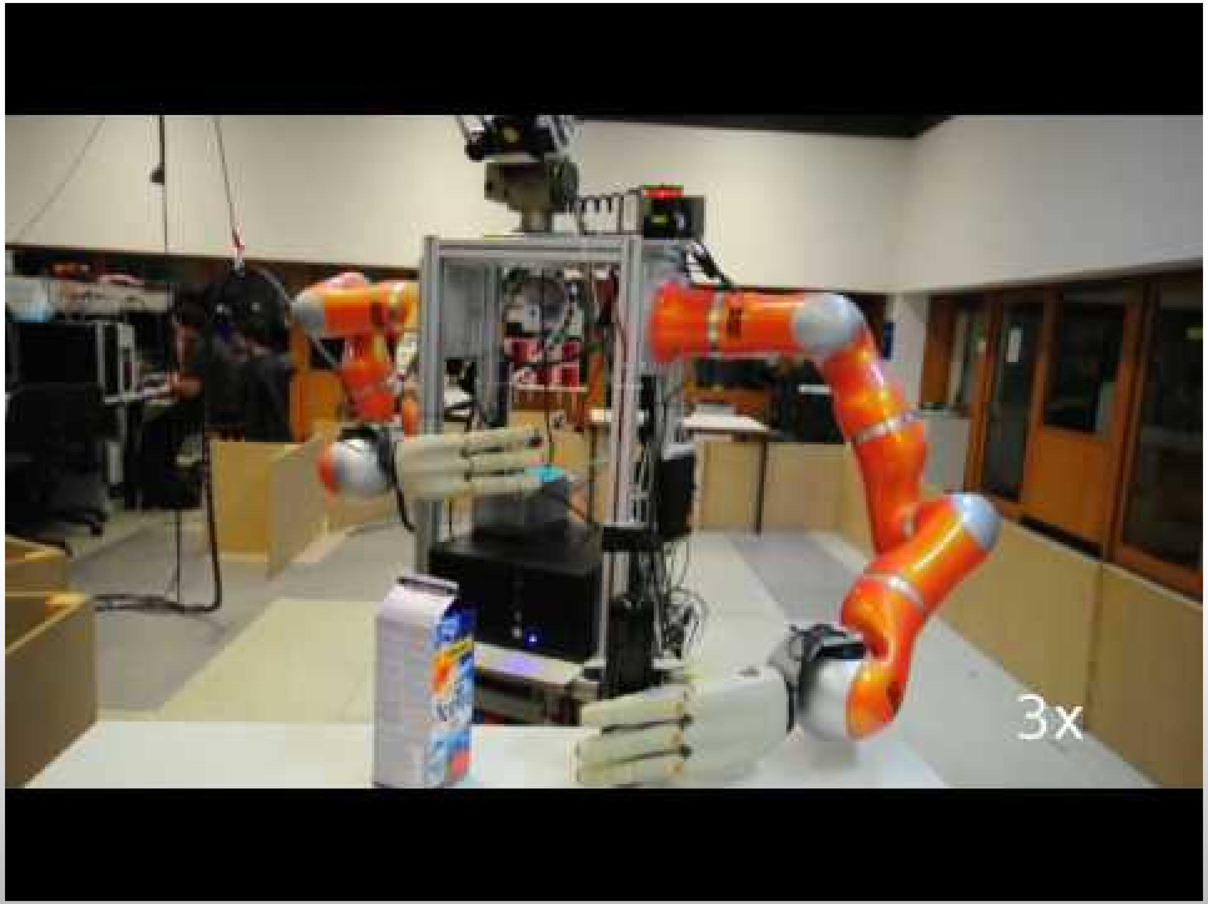
- Configuring arm-navigation for your robot
- Understanding what components are available, what they do, and how they fit together
- Learning about all the support systems that are required to transition from motion planners to producing, visualizing, and executing trajectories

Robot URDF

- Robot name
- Joints
  - Single DOF
    - Prismatic
    - Revolute
  - Fixed
  - Joint limits (if has them)
    - Physical
    - Safety
- Links
  - Positions relative to joints
  - Geometry (meshes or primitives)
    - Visual
    - Collision

TUM-Rosie

iros_tutorial_resources/robot_defs/ias_robot_defs
urdf_package:=ias_robot_defs
urdf_path:=robots/rosie.expanded.xml



Robot Kinematic Tree

- World joint (odom_combined) -> Root link (base_footprint)
  - Supplied by external system (not part of URDF)
- Root link -> child joints (one->many)
- Child joint -> child link (one->one)
- Forward kinematics computes link positions given joint positions

Planning Groups (1/2)

- Kinematics Chains (at least one per robot)
  - Base link
    - Anchors robot
    - Not part of planning group
  - Tip link
    - Child link of last joint
    - Is part of planning group
  - Groups joints from child of base link to parent of tip link
    - And any fixed joints
  - Group links all children of group joints
  - Updated links all group links and everything further down in the tree

Planning Groups (2/2)

- Joint Collections (optional)
  - Arbitrary collection of joints
  - Useful for defining end effector groups
  - Group links all children of joints
  - Updated links everything further down the kinematic tree from any group joint

Self-collision Operations

- Pair classes disabled by default
  - Adjacent links in tree
  - Link pairs always in collision
  - Link pairs often in collision (>50% of samples)
  - Link pairs in collision in default state
    - 0.0 for joint position if valid
    - (upper_bound-lower_bound)/2.0 otherwise
  - Link pairs never in collision
- Randomized sampling strategy to differentiate between never and sometimes
  - More disables means more efficiency
  - Even if we're wrong on some, generally ok

Auto-generated files

- in <your_robot_name>_arm_navigation/config/
  - <robot_name>_planning_description.yaml
    - multi_dof_joints - world_joint
    - groups
    - collision_operations
  - joint_limits.yaml
    - Extra velocity/acceleration limits
  - ompl_planning.yaml
    - Planner configuration
    - Group configuration
- in <your_robot_name>_arm_navigation/launch/
  - component launch files
  - move_<group_name>.launch
  - <your_robot_name>_arm_navigation.launch

Planning components

- Inverse kinematics
  - KDL-based numerical solver
- Planner
  - OMPL
- Trajectory filter
  - Cubic-spline shortcutter
- Sub-components (C++ classes)
  - Forward kinematics
  - Collision checking
    - Unpadded self-collision checking
    - Padded environment collision checking
  - State and trajectory validity
- Planning components visualizer

Running with a real or simulated robot

- There is a state of the world
  - Current robot configuration
  - collision_map
  - Recognized objects
- Monitoring system provides this current state
  - You can provide a diff
- Running move_<group_name>.launch
  - Communicates with monitors and controllers
  - Implements a state machine
    - Call collision-aware IK (if necessary)
    - Call planner to produce trajectory
    - Call trajectory filter on planner trajectory
    - Pass to controller and monitor result

Warehouse viewer

- Right now:
  - Not just ROS messages, but also metadata
  - Build up and save interesting worlds, requests, trajectories
    - Automatic logging from components
  - Execute and record trajectories
    - Running alongside simulation or a real robot
  - Can implement heavily human-aided manipulation
- In the future:
  - Easily swap and configure components
  - Collect metrics for different components across large datasets with different robots
  - Include general manipulation functionality
    - Grasp point generation/evaluation
    - Grasp pipeline scripting

Take-homes:

- You can get all of this working for your robot
    - And we can help!
- Two primary points of entry into system
    - Developing applications on top of move_<group_name> and existing components
    - Developing more powerful components
        - Plenty of supporting infrastructure
        - Interaction/metrics capturing system coming soon
- There's still more to do to get executed collision-free plans in the real world
    - Perception
    - Controllers