MoveIt and the rest of ROS:
Perception, Control, and Simulation

ROSWorld October 2021 - Mobile Manipulation Workshop

Vatan Aksoy Tezer
@vatanaksoytezer
Outline

- A brief Overview of ROS Communication
- How does a mobile manipulator work, how different pipelines connect?
- Navigation
- Perception
- Simulation and Control
A Brief Overview of ROS Communication
Pub/Sub Basics

Pub/Sub Topics, Services and Actions are the primary communication patterns used in ROS and also MoveIt!

Helpful command line tools:

- rostopic list
- rostopic echo <topic_name>
- rostopic hz <topic_name>
- ros2 topic list
- ros2 topic echo
...

Understanding ROS2 topics
Remembering Actions
MoveIt Uses Actions to send the robot / simulation planned joint trajectories.

Typically MoveIt is the client and server is opened up by ros(2)_control.

You can also use your own FollowJointTrajectory server as well!

```python
from control_msgs.action import FollowJointTrajectory
from trajectory_msgs.msg import JointTrajectory
import rclpy
from rclpy.action import ActionServer, Server
from rclpy.node import Node
from math import sqrt

class FollowJointTrajectoryActionServer(Node):
    def __init__(self):
        super().__init__('_follow_joint_trajectory_action_server')
        self.joint_trajectory_publisher = self.create_publisher(JointTrajectory, '/joint_trajectory', 10)
        self.action_server = ActionServer(self,
            FollowJointTrajectory,
            '/stretch_controller/follow_joint_trajectory',
            self.execute_callback)

    def execute_callback(self, self, goal_handle: ServerGoalHandle):
        self.get_logger().info('Executing goal...')
        result = FollowJointTrajectory.Result()
        trajectory = goal_handle.request.trajectory # type: JointTrajectory
        self.joint_trajectory_publisher.publish(trajectory)
        goal_handle.succeed()
        return result

def main(args=None):
    rclpy.init(args=args)
    follow_joint_trajectory_action_server = FollowJointTrajectoryActionServer()
    rclpy.spin(follow_joint_trajectory_action_server)

if __name__ == '__main__':
    main()
```
How does a mobile manipulator work, how different pipelines connect?
Mobile Manipulation Work Cycle

Credit: ETH Zurich Mobile Robot Lecture Notes: https://asl.ethz.ch/education/lectures/autonomous_mobile_robots.html
Perception
MoveIt Calibration

Perform hand-eye calibrations in RViz

Generate a target image to print

Move arm, acquire 5+ sample poses

Export EEF->camera transform
Octomap and Collision Awareness

3D occupancy map for collision checking

Update from depth map or point cloud
Navigation
Mobile Base Planning in MoveIt 2

- Holonomic movements were supported for a long time (quadcopters, holonomic robots)
- Stretch is a differential drive robot so PickNik has added a new differential drive motion model to the planar joints.
- Basically a virtual joint defined in SRDF that publishes valid cmd_vel commands.
- `<virtual_joint name="position" type="planar" parent_frame="odom" child_link="base_link"/>
  <joint_property joint_name="position" property_name="motion_model" value="diff_drive"/>
  <joint_property joint_name="position" property_name="min translational distance" value="0.05"/>`
Using nav2 with MoveIt 2

- MoveIt 2 perception pipeline support PointCloud messages that LIDARs usually publish.
- But, in navigation we might want to take more than pointclouds: such as inflation layer.
- MoveIt 2 does not support navigation layers and is not meant to be a replacement for navigation2.
- Using nav2_simple_commander we can give the MoveIt generated waypoints and let nav2 execute the trajectory for us instead of MoveIt if desired.
Using nav2 with MoveIt 2

```python
def execute_callback(self, goal_handle: server.ServerGoalHandle):
    self.get_logger().info('Executing goal...
    result = FollowJointTrajectory.Result()
    trajectory = goal_handle.request.trajectory # type: JointTrajectory
    self._joint_trajectory_publisher.publish(trajectory)
    multidof_trajectory = goal_handle.request.multi_dof_trajectory # type: MultiDOFJointTrajectory
    goal_pose = PoseStamped()
    goal_pose.header.frame_id = "odom"
    goal_pose.header.stamp = self.get_clock().now().to_msg()
    goal_pose.pose.position.x = multidof_trajectory.points[-1].transforms[0].translation.x
    goal_pose.pose.position.y = multidof_trajectory.points[-1].transforms[0].translation.y
    goal_pose.pose.position.z = multidof_trajectory.points[-1].transforms[0].translation.z
    goal_pose.pose.orientation.x = multidof_trajectory.points[-1].transforms[0].rotation.x
    goal_pose.pose.orientation.y = multidof_trajectory.points[-1].transforms[0].rotation.y
    goal_pose.pose.orientation.z = multidof_trajectory.points[-1].transforms[0].rotation.z
    goal_pose.pose.orientation.w = multidof_trajectory.points[-1].transforms[0].rotation.w
    print("Target pose:", goal_pose.pose)
    self.nav.goToPose(goal_pose)
    goal_handle.succeed()
    return result
```
Simulation and Control
Stretch's Sensors

- **Head**
  - Intel Realsense D435i
  - 4 channel ReSpeaker MicArray V2.0
  - 8W stereo audio out w/volume adjust
- **Base**
  - RPLidar A1
  - 9 DoF IMU
- **Wrist**
  - 3 DoF Accelerometer
- **Force sensing (via motor current) on arm and lift**
- **Aruco tags**
Stretch's Sensors

- Most of the sensors are ported and working quite well in Ignition Gazebo.
- Use gazebo and sensor tags, same with ROS and Gazebo Classic.
- Realsense, LIDAR and all the IMUs are simulated.
- Force sensing not implemented yet but now possible with Ignition Fortress!
- ReSpeaker MicArray not simulated but possible through a ROS2 node that connects to simulator PC's microphone and speaker.
Simulating and Controlling Stretch in Ignition Gazebo and ROS 2

Here is what changed since ROS1 and Gazebo Classic:

- You can still use xacros or urdfs
- You can still use gazebo tags (sensor etc.) in xacros (see [http://sdformat.org/spec?elem=sensor&ver=1.8](http://sdformat.org/spec?elem=sensor&ver=1.8) for sensor parameters)
- You might need to add some system plugins to your world file
- Joint trajectory plugin is great to control joints, joint states plugin is great publishing joint states until ign_ros2_control is here [https://github.com/ignitionrobotics/ign_ros2_control/pull/1](https://github.com/ignitionrobotics/ign_ros2_control/pull/1).
- Tune gains from joint_trajectory plugin until ros2_control arrives.
- ign_ros_bridge for ROS <-> IGN communication
- Porting existing worlds is super easy to Ignition (See aws_robomaker_small_house in action!)
Putting Everything Together with Guided Exploration!
Thanks!

PickNik Robotics
picknik.ai
Colorado, USA
@picknikrobotics

Vatan Aksoy Tezer
vatan@picknik.ai