UiO : University of Oslo

IN3140
Open-Source Robotics

IN3140 - Introduction to Robot Operating System (ROS): Part II

Abbas Tariverdi
abbast@uio.no
Spring 2021

Bekkeng 09.04.2021

Recap of the previous lecture

- What is ROS?
- Concepts: Nodes, Messages, Topics, Services, roscore(ROS Master)
- RQT Tools: rqt_plot and rqt_graph.

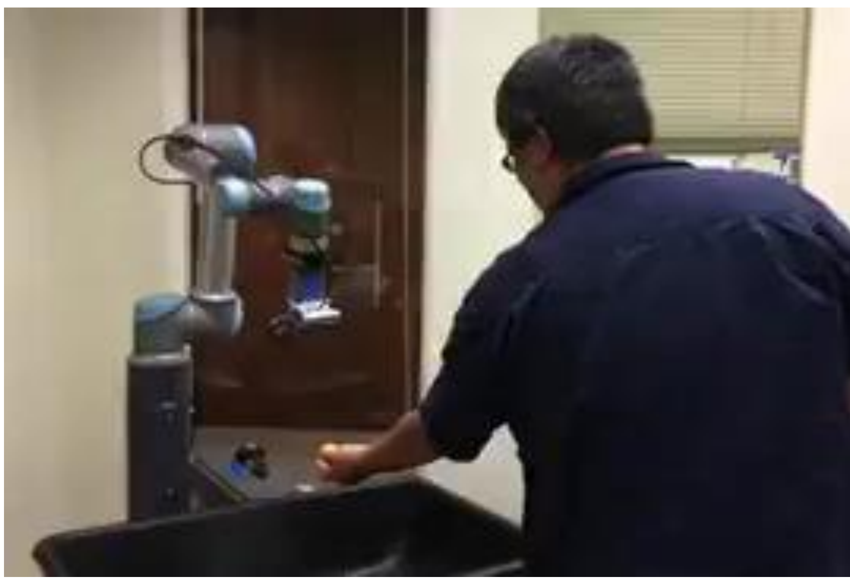
Hands-on

- Setting up a new ROS Installation
- Creating workspace
- Creating packages
- Working with Nodes, Topics, Messeges
- Simple Subscriber/Publisher

1. Tutorials On GitHub
2. http://wiki.ros.org/ROS/Tutorials

Main Tasks in Robotics

- Motion/Trajectory Planning (Manipulator kinematics: Forward and Inverse Kinemtaics)- Collision/obstacle avoidance
- Control (Position and Force Control)



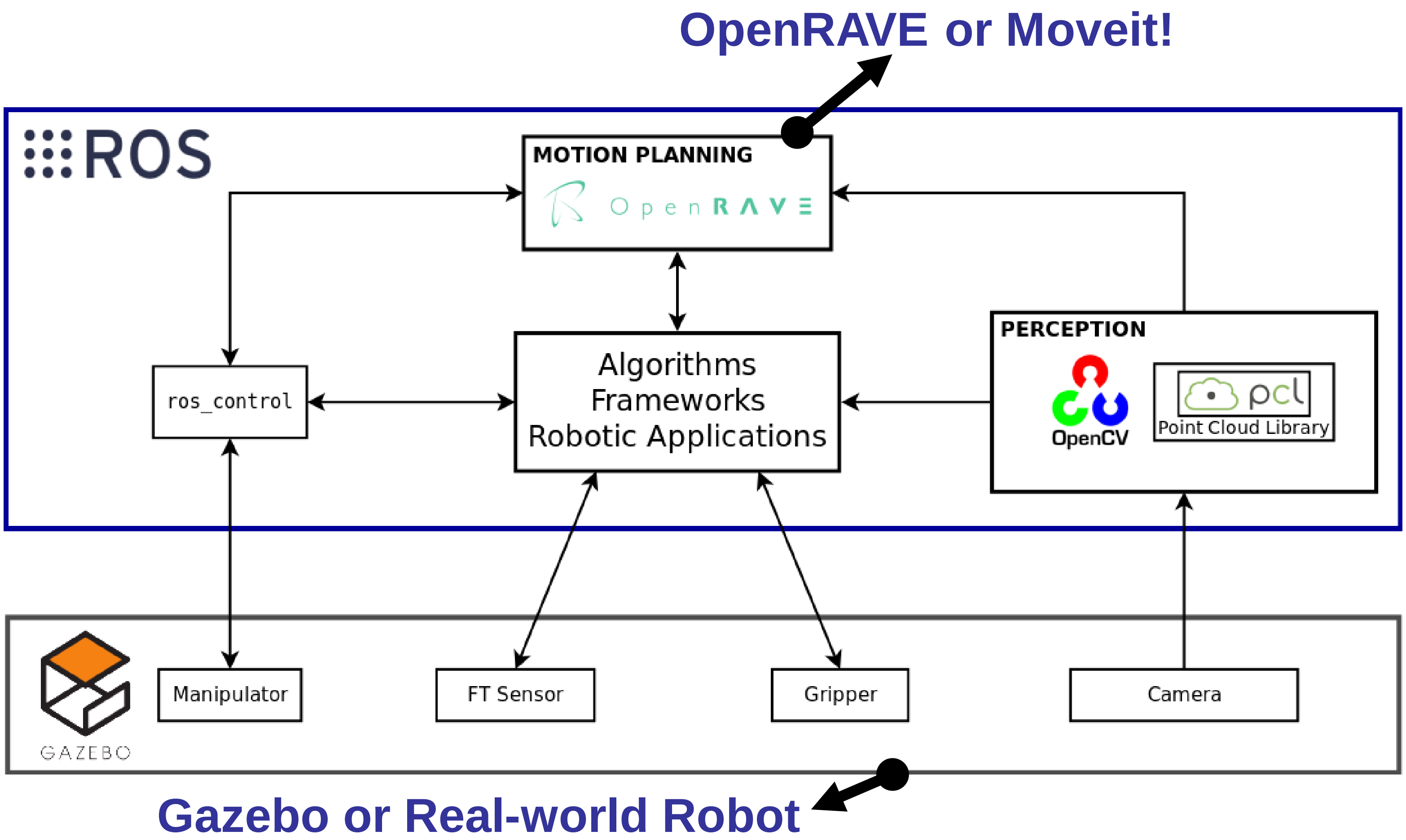
https://andyzeng.github.io
https://www.ros.org/

Lecture Plan

Going through

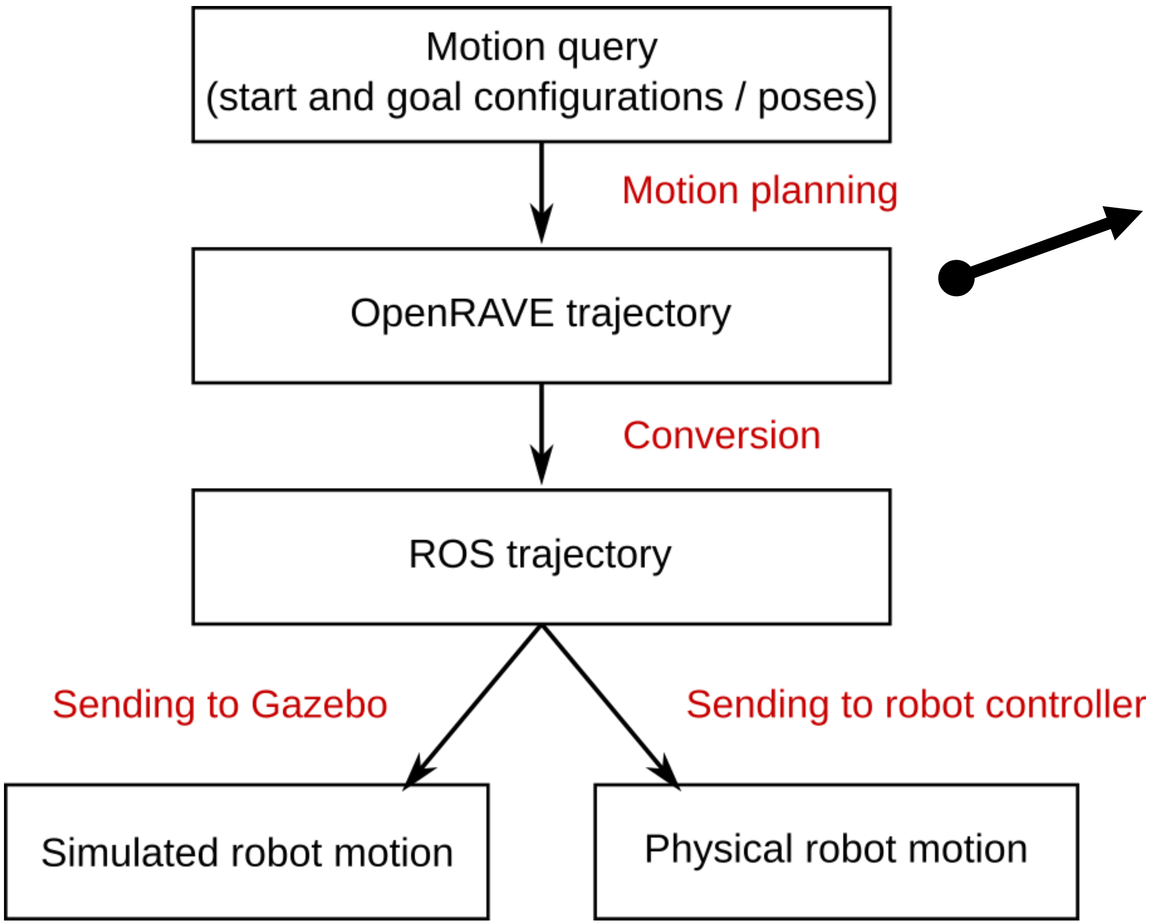
- Movieit!
- Gazebo
- Integration
- ROS-Industrial: Universal Robot (UR5) & ROS Control

Implementation a Robotic Tasks



http://osrobotics.org/osr/

Overview of Motion Planning Procedure

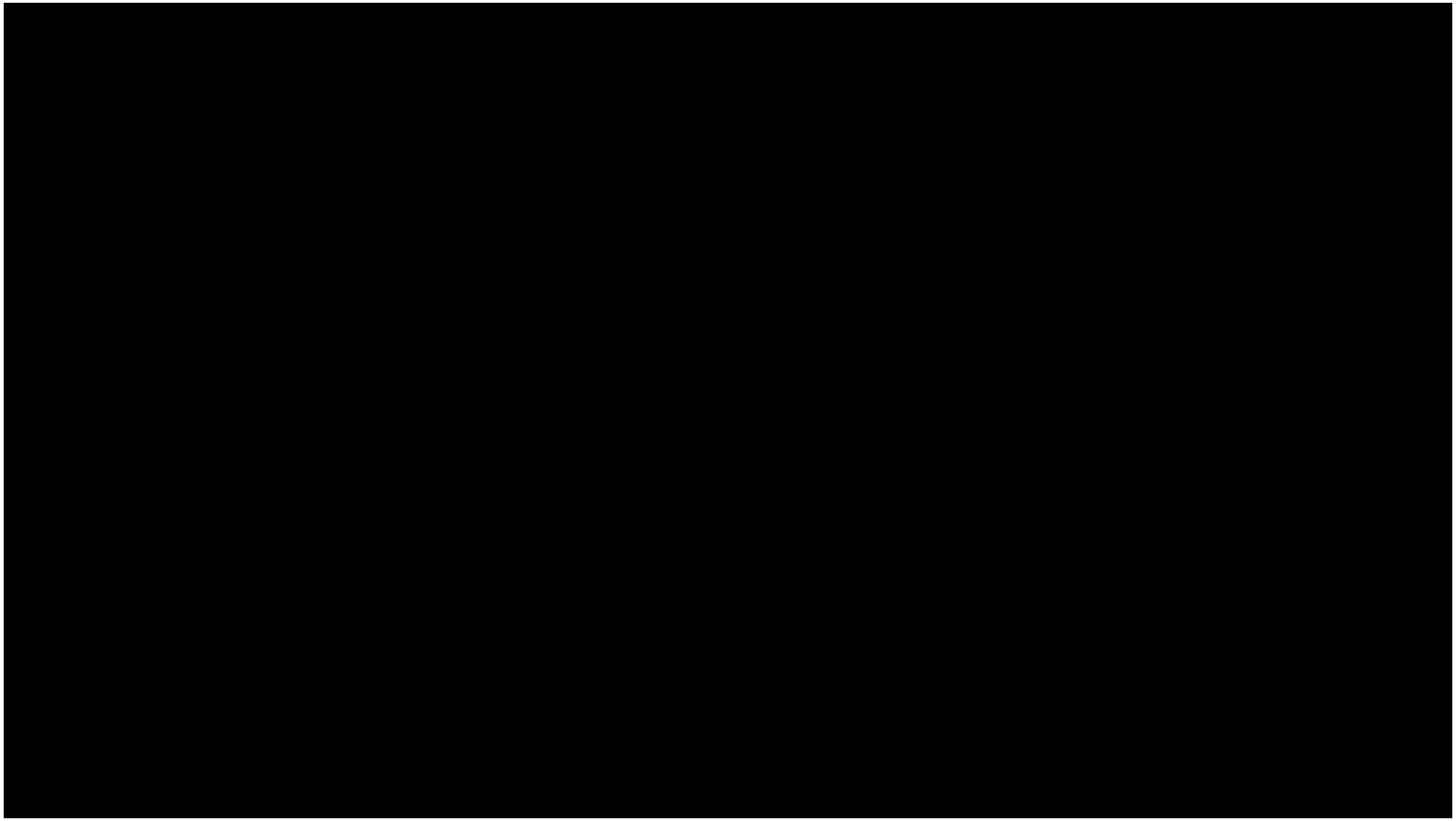


OpenRAVE
Moveit!

Review of Technical Capabilities

https://moveit.ros.org

MoveIt



MoveIt Motion Planning

MoveIt! includes a variety of robust and state-of-the-art motion planners:

- Sampling-based motion planning algorithms (OMPL)
- Covariant Hamiltonian optimization for motion planning (CHOMP)
- Stochastic Trajectory Optimization for Motion Planning (STOMP)
- TrajOpt is a sequential convex optimization algorithm

MoveIt Constraints

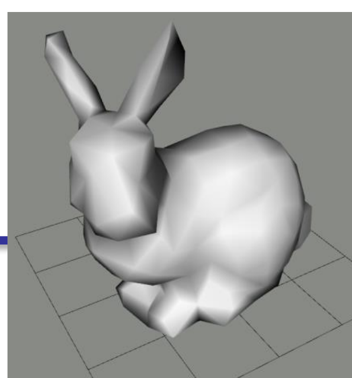
You can specify the following kinematic constraints:

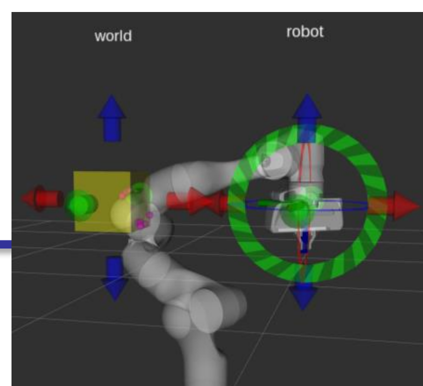
- **Position constraints** – restrict the position of a link to lie within a region of space
- **Orientation constraints** – restrict the orientation of a link to lie within specified roll, pitch or yaw limits
- **Visibility constraints** – restrict a point on a link to lie within the visibility cone for a particular sensor
- **Joint constraints** – restrict a joint to lie between two values
- **User-specified constraints** – you can also specify your own constraints with a user-defined callback.

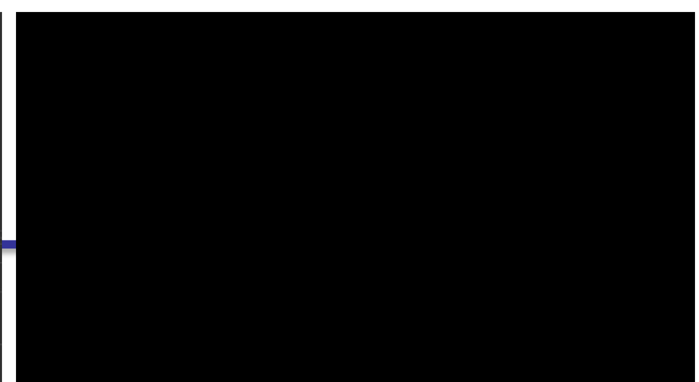
MoveIt Scene Collision Objects

You can specify the following kinematic constraints:

- static objects (objects rigidly fixed on the robot workspace)
- dynamic objects (objects with which the robot can interact, i.g. pick, place, push ...etc)
- Moveit Collision Objects published through moveit_msgs/CollisionObject messages
- mesh (.stl or .dae) or primitive objects (Boxes, Spheres, Cylinders, and Cones), OctoMap



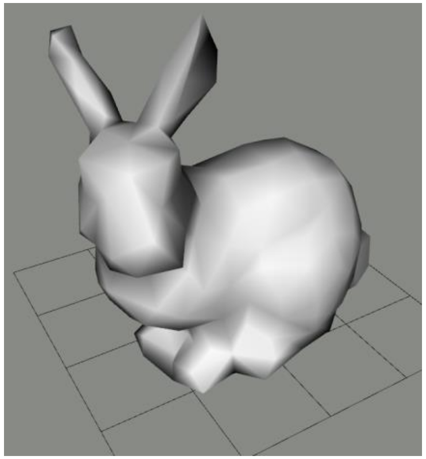


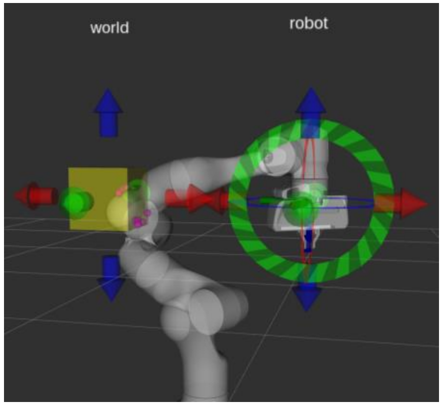


MoveIt Scene Collision Objects

Collision Objects:

- mesh (.stl or .dae) or primitive objects
- (Boxes, Spheres, Cylinders, and Cones), OctoMap





MoveIt How to Use it?!

To simulate and play around with Universal Robot UR5:

- Have ROS installed.
- Create a work-space: `mkdir -p ~/ws_moveit/src`
- From ROS-Industrial GitHub Page:

```
git clone -b melodic-devel https://github.com/ros-industrial/universal_robot
```

- Install any new dependencies that may be missing:

```
rosdep install -y --from-paths . --ignore-src --rosdistro noetic
```

- Re-build and re-source the workspace and enjoy:

`catkin_make` and `source devel/setup.bash`

```
roslaunch ur5_moveit_config moveit_rviz.launch
```

ros-planning.github.io/moveit_tutorials/doc/realtime_servo/realtime_servo_tutorial.html?highlight=ur5

GAZEBO

GAZEBO

Review of Technical Capabilities

http://gazebosim.org/

GAZEBO

simulation using Gazebo within a ROS environment:

- **Gazebo basics**: understanding the Gazebo simulation infrastructure
- **Integration to ROS**: understanding how Gazebo is integrated within ROS by means of the gazebo_ros package
- **Configuring launch files**
- **Modeling robots for Gazebo**

https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

GAZEBO

simulation using Gazebo within a ROS environment:

- **Gazebo basics**: understanding the Gazebo simulation infrastructure
- **Integration to ROS**: understanding how Gazebo is integrated within ROS by means of the gazebo_ros package
- **Configuring launch files**
- **Modeling robots for Gazebo**

https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

GAZEBO Gazebo basics, Gazebo files

To run a Gazebo simulation you need:

- **A world file**: A file with extension `.world` that contains all the elements in a simulation, including robots, lights, sensors, and static objects, formatted using the Simulation Description Format (SDF). Some world files can be found at `/usr/share/gazebo-9/worlds`).

https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

GAZEBO Gazebo basics, Gazebo files

To run a Gazebo simulation you need:

- **Model files**: SDF files used to describe objects and robots (a single `<model> … </model>`). Models are included in world files using the include tag:

```
<include> <uri>model://model_file_name</uri> </include>
```

The components of a model are:

- **Links**: A link contains the physical properties of one body of the model.
- **Joints**: A joint connects two links.

https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

GAZEBO Gazebo plugins

A plugin is a chunk of code that is compiled as a shared library and inserted into the simulation. There are currently 6 types of plugins:

- **World**: Attached to the world to control world properties.
- **Model**: Attached to a model to control the joints and the state.
- **Sensor**: Attached to a sensor to acquire sensor information and control sensor properties.
- **Visual**: A plugin to access the visual rendering functions.

https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

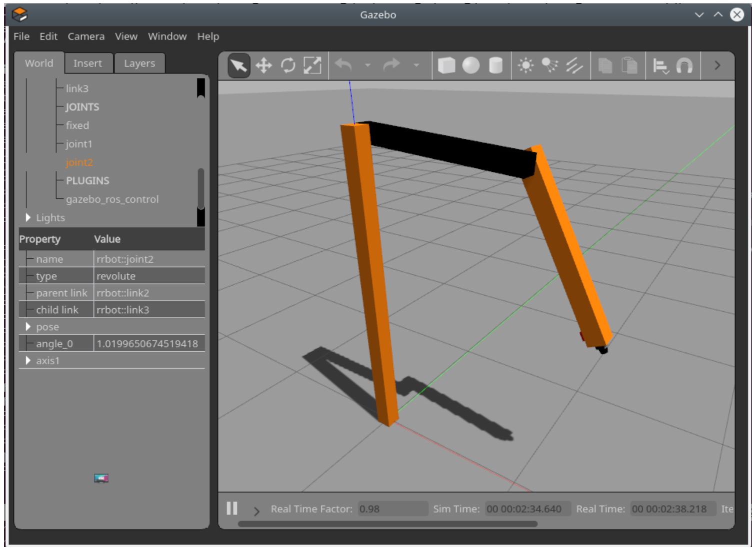
GAZEBO rrbot example

RRBot, or "Revolute-Revolute Manipulator Robot", is a simple 3-linkage, 2-joint arm.

```
cd ~/catkin_ws/src/
git clone https://github.com/ros-simulation/gazebo_ros_demos.git
cd ..
catkin_make

rosed rrbot_description rrbot.xacro

roslaunch rrbot_gazebo rrbot_world.launch
```



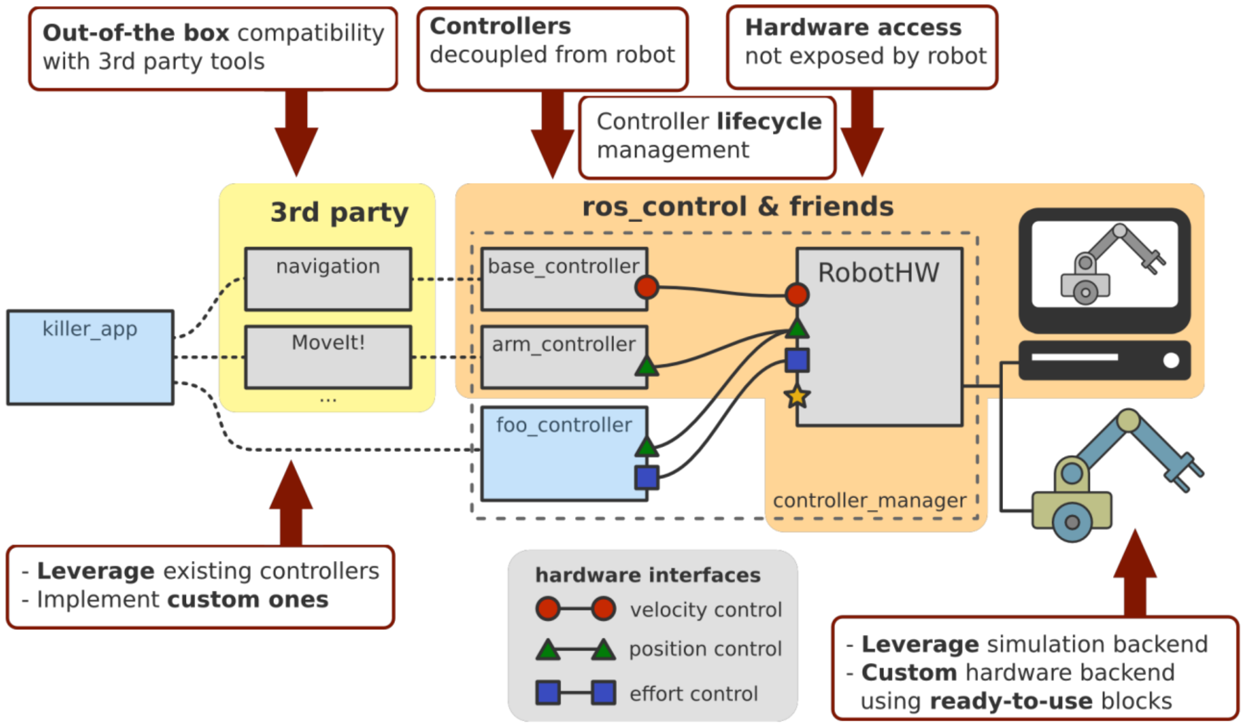
https://sir.upc.edu/projects/rostutorials/8-gazebo_basics_tutorial/index.html#basics-label

http://gazebosim.org/tutorials/?tut=ros_urdf#Sharingyourrobotwiththeworld

Robot Control: ros_control overview

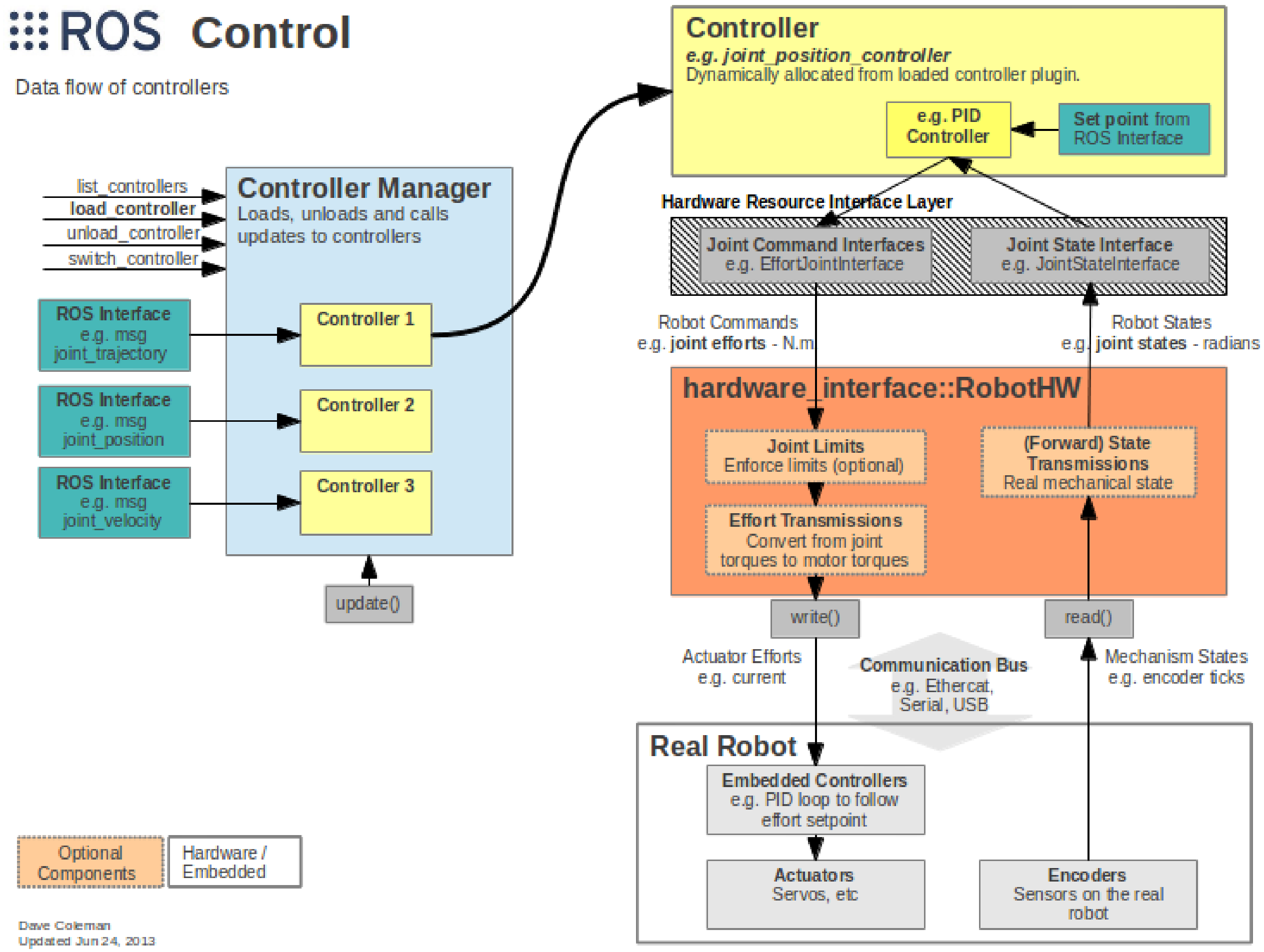
Understand the structure of the `ros_control` framework.

Available controllers and concepts.



https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

Robot Control: ros_control overview



https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

ROS Control: Available Controllers

The main ROS controllers are grouped according to the commands get passed to your hardware/simulator:

- `effort_controller`: efforts commands are used to control joint positions, velocities or efforts.
- `position_controllers`: position commands are used to control joint positions.
- `velocity_controllers`: velocity commands are used to control joint velocities.

https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

Configuring and launching controllers

Controllers are usually defined with `yaml` files

```yaml
rrbot:
  # Publish all joint states -----------------------------------
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  # Position Controllers ---------------------------------------
  joint1_position_controller:
    type: effort_controllers/JointPositionController
    joint: joint1
    pid: {p: 100.0, i: 0.01, d: 10.0}
  joint2_position_controller:
    type: effort_controllers/JointPositionController
    joint: joint2
    pid: {p: 100.0, i: 0.01, d: 10.0}
```

https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

Gazebo and ROS Control

- Run the simulation

```
roslaunch rrbot_gazebo rrbot_world.launch
roslaunch rrbot_control rrbot_control.launch
```

- Manually send example commands

```
rostopic pub -1 /rrbot/joint1_position_controller/command std_msgs/Float64 "data: 1.5"
rostopic pub -1 /rrbot/joint2_position_controller/command std_msgs/Float64 "data: 1.0"
```

- Use RQT To Send Commands

```
rosrun rqt_gui rqt_gui
```

https://sir.upc.edu/projects/rostutorials/10-gazebo_control_tutorial/index.html

Thanks for your attention!

Any Question?