ROS-Industrial Basic Developer’s Training Class

October 2021

Southwest Research Institute

Session 3:

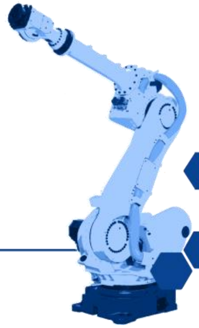
Motion Control of Manipulators

Southwest Research Institute

Outline

- ROS1 Intro
- URDF
- TF
- Motion Planning in ROS

ROS-1 Intro



ROS-1/2 Transition

- ROS community is currently in transition
  - most core packages and features are in ROS2
  - many other packages are still only in ROS1
    - hardware drivers (cameras, robots)
    - algorithms (perception, motion planners)
- ROS1 and ROS2 systems can’t interact directly
  - ROS provides a ros_bridge node to help
- Many projects will continue to use hybrid ROS1/2 systems in the near future.

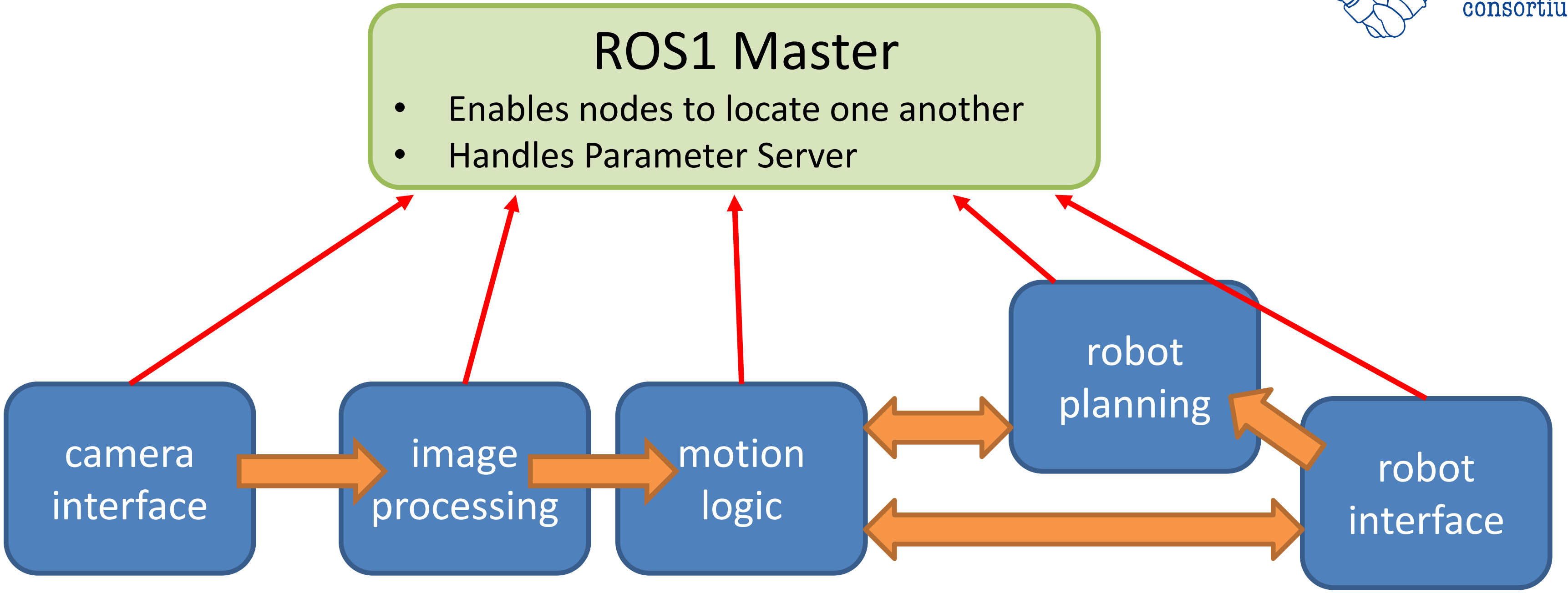
ROS1 Scope

Since most **new development** will be in ROS2, this section focuses on **runtime differences** - build, execution, and command-line tools.

ROS1/2 Major Differences

	ROS1	ROS2
Comms Protocol	XMLRPC + TCPROS	DDS
Architecture	ROS Master + Distributed	Fully Decentralized
Build System	catkin (cmake-based)	colcon / ament (cmake-based)
Build Output	ros1_ws/devel	ros2_ws/install
Parameters	Global Parameter Server Dynamic Reconfigure	Per-Node Parameters
Launch	XML	Python (+XML, YAML alternatives)
Commands	roslaunch, rosrun, rospack, rostopic, …	ros2 launch, ros2 run, ros2 pkg, ros2 topic
Platforms	Primarily Ubuntu	Linux, MacOS, Windows

ROS1 Master

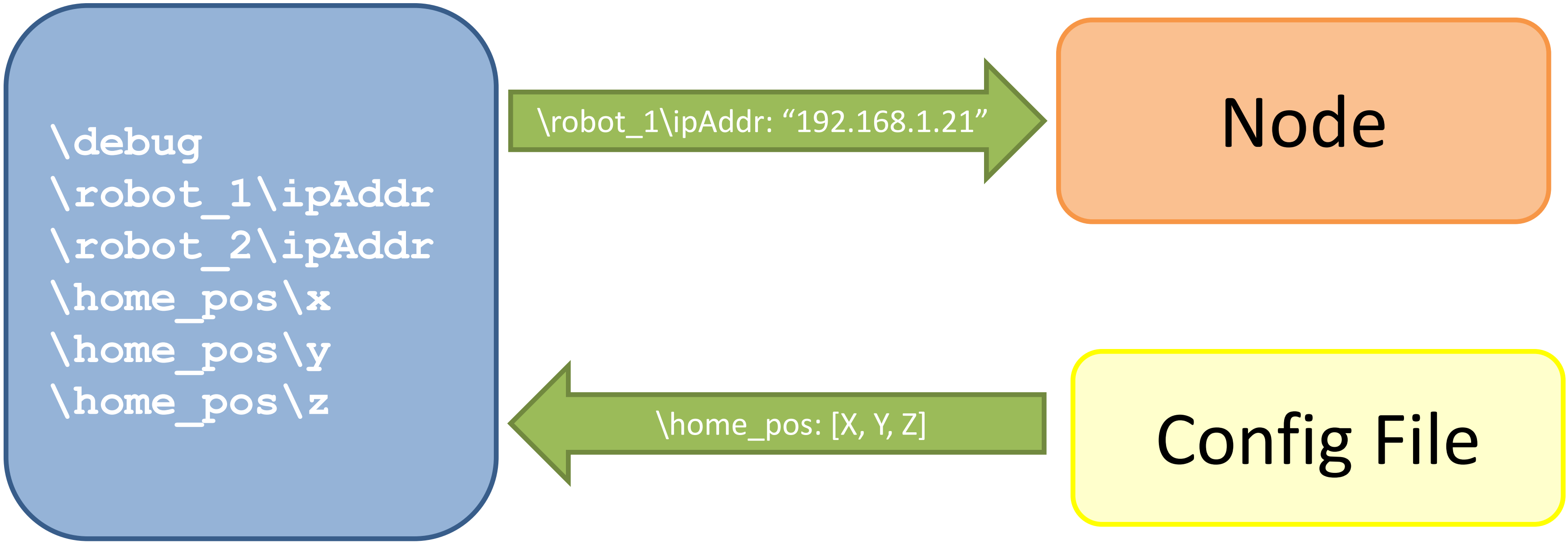


- Each ROS1 System must have a **single** master
- Start with: roscore or roslaunch

ROS1 Parameters

ROS1 Parameters are like **Global Data**

Parameter Server



```
\debug
\robot_1\ipAddr
\robot_2\ipAddr
\home_pos\x
\home_pos\y
\home_pos\z
```

\robot_1\ipAddr: “192.168.1.21”

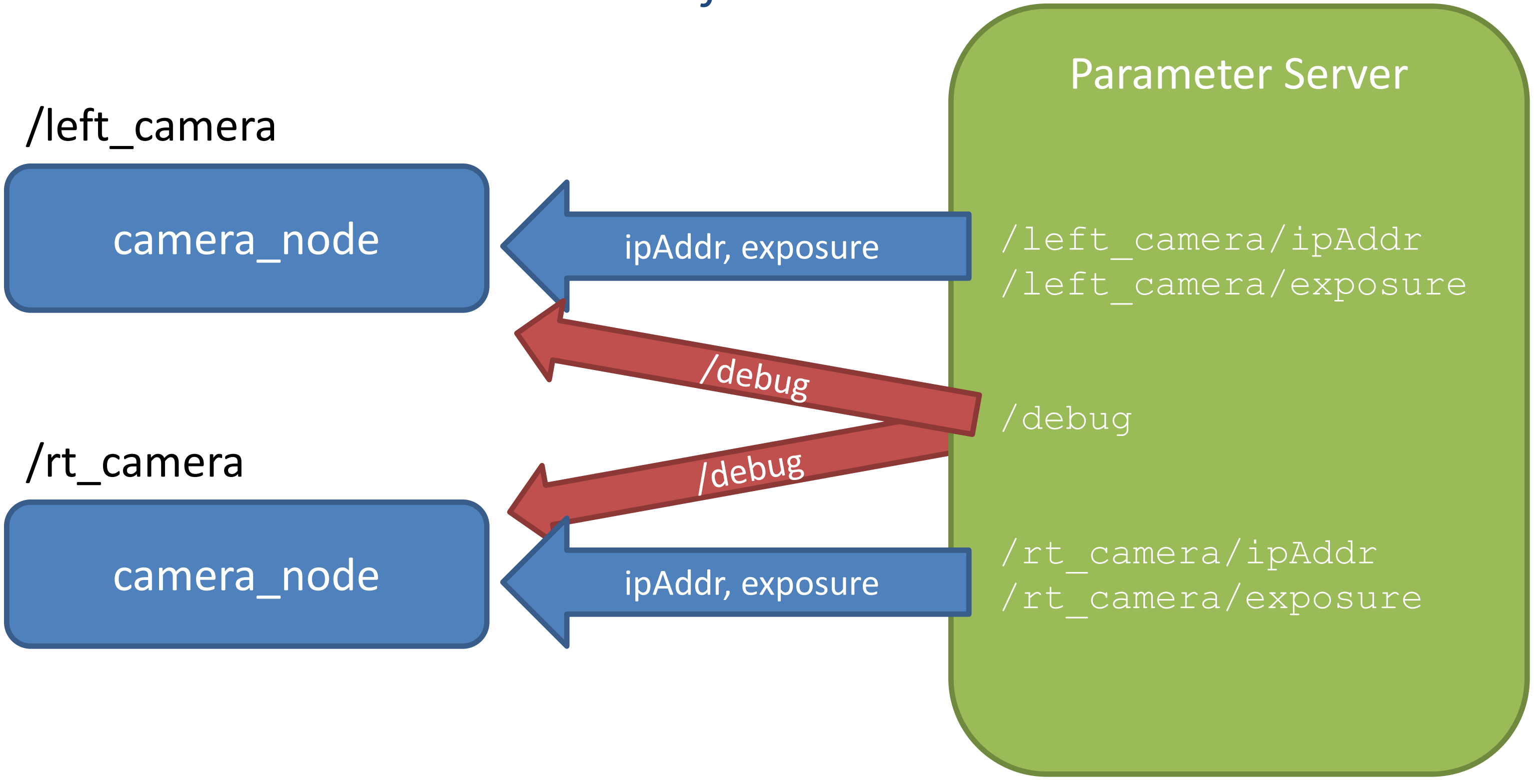
Node

\home_pos: [X, Y, Z]

Config File

ROS1 Parameter Namespaces

- Folder Hierarchy allows Separation:
  - *Separate nodes can co-exist, in different "namespaces"*
  - *relative vs. absolute name references*



ROS1 Launch Files (XML)

- **`<launch>`** – Required outer tag
- **`<rosparam>`** or **`<param>`** – Set parameter values
  - *including load from file (YAML)*
- **`<node>`** – start running a new node
- **`<include>`** – import another launch file

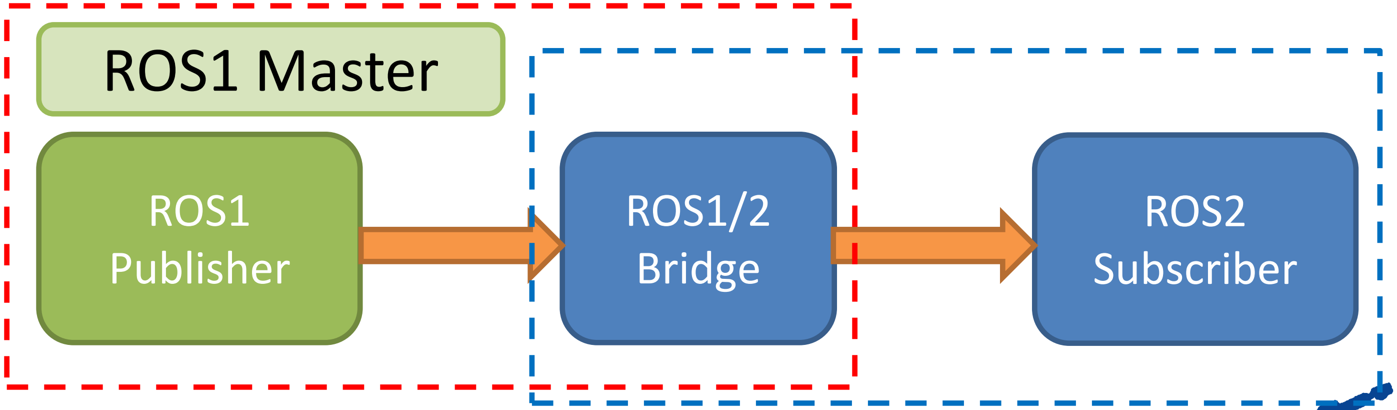
```
<launch>
  <rosparam param=“/robot/ip_addr“>192.168.1.50</rosparam>

  <param name="robot_description“ textfile="$(find robot_pkg)/urdf/robot.urdf"/>

  <node name=“camera_1" pkg=“camera_aravis” type=“camnode" />

  <node name=“camera_2" pkg=“camera_aravis“ type=“camnode“ />

  <include file=“$(find robot_pkg)/launch/start_robot.launch” />
</launch>
```

ROS1 Common Commands

- Build
  - `catkin build`
- Run
  - `roscore`
  - `rosrun mypackage mynode`
  - `roslaunch mypackage mylaunch.launch`
- Inspect
  - `rospack find mypackage`
  - `rostopic list  (+ rostopic echo)`

ROS1/2 Bridge

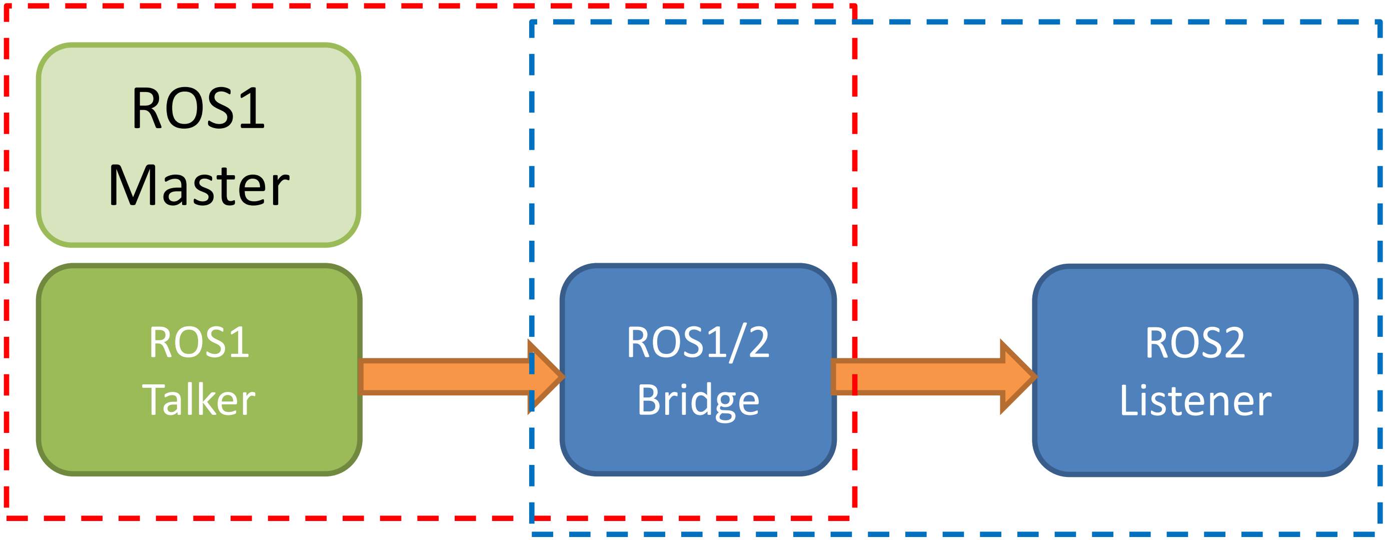
- ROS1 and ROS2 Systems must be **separate**
  - different workspaces, different terminals
  - ROS1 nodes can't talk directly to ROS2 nodes
- ros_bridge provides mapping between ROS1/2 topics, services, and actions
  - It must be recompiled to add support for new msg types.



Exercise 3.0a

ROS1 Basics

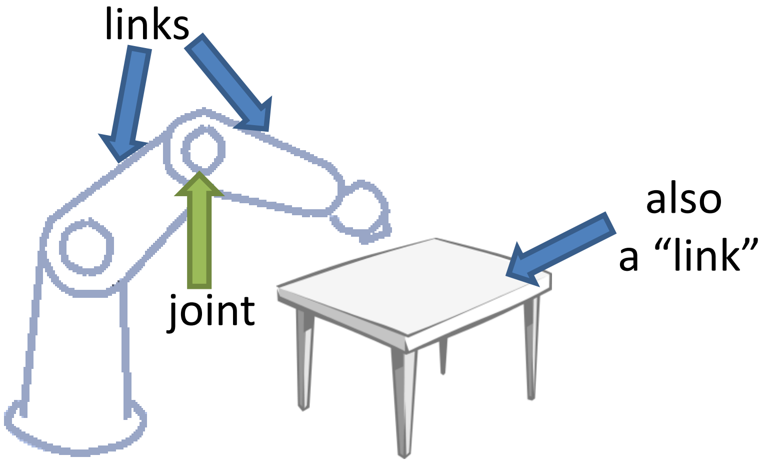
Intro to ROS1 Bridge



URDF:
Unified Robot Description Format

URDF: Overview

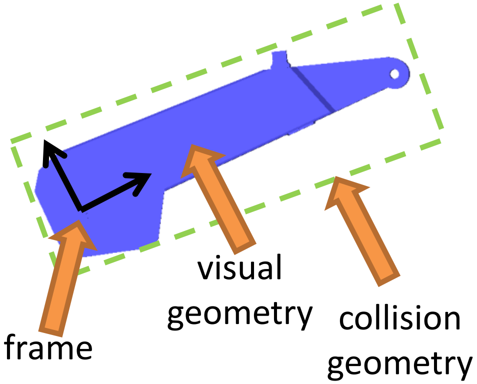
- URDF is an **XML**-formatted file containing:
  - **Links** : coordinate frames and associated geometry
  - **Joints :** connections between links
- Similar to DH-parameters (but way less painful)
- Can describe entire workspace, not just robots



URDF: Link

- A **Link** describes a physical or virtual object
  - Physical: robot link, workpiece, end-effector, ...
  - Virtual : TCP, robot base frame, ...
- Each link becomes a TF frame
- Can contain visual/collision geometry [optional]
- http://wiki.ros.org/urdf/XML/link

```
<link name=“link_4“>
   <visual>
        <geometry>
             <mesh filename=“link_4.stl"/>
        </geometry>
        <origin xyz="0 0 0" rpy="0 0 0" />
   </visual>
   <collision>
        <geometry>
             <cylinder length="0.5" radius="0.1"/>
        </geometry>
        <origin xyz="0 0 -0.05" rpy="0 0 0" />
   </collision>
</link>
```



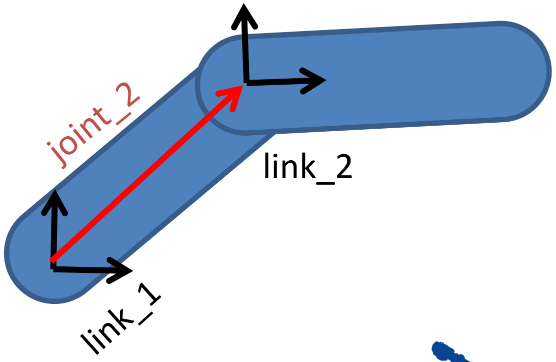
URDF Transforms

X/Y/Z	Roll/Pitch/Yaw
Meters	Radians

URDF: Joint

- A **Joint** connects two **Links**
  - Defines a **transform** between **parent** and **child** frames
    - Types: *fixed, free, linear, rotary*
  - Denotes axis of movement *(for linear / rotary)*
  - Contains joint limits on position and velocity
- ROS-I conventions
  - X-axis front, Z-Axis up
  - Keep all frames similarly rotated when possible
- http://wiki.ros.org/urdf/XML/joint

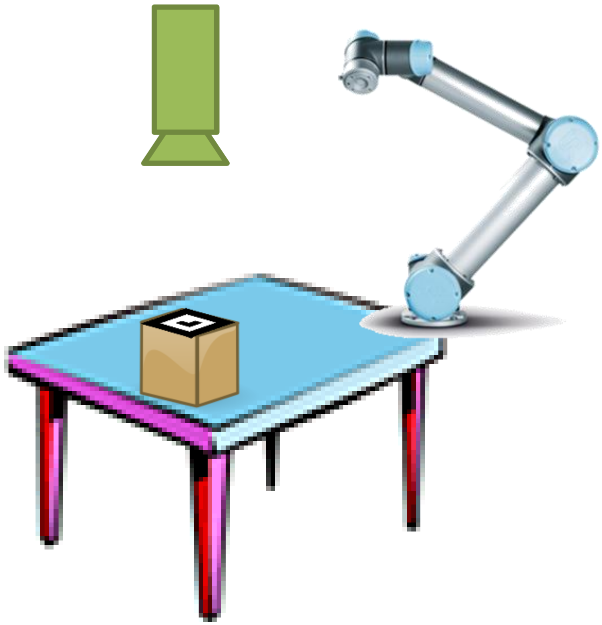
```
<joint name=“joint_2" type="revolute“>
   <parent link=“link_1"/>
   <child link=“link_2"/>
   <origin xyz=“0.2 0.2 0" rpy="0 0 0"/>
   <axis xyz=“0 0 1"/>
   <limit lower="-3.14" upper="3.14" velocity="1.0"/>
</joint>
```

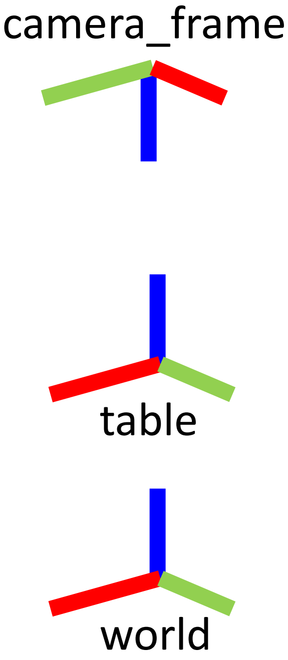


Exercise 3.0

Exercise 3.0

Create a simple urdf





URDF: XACRO

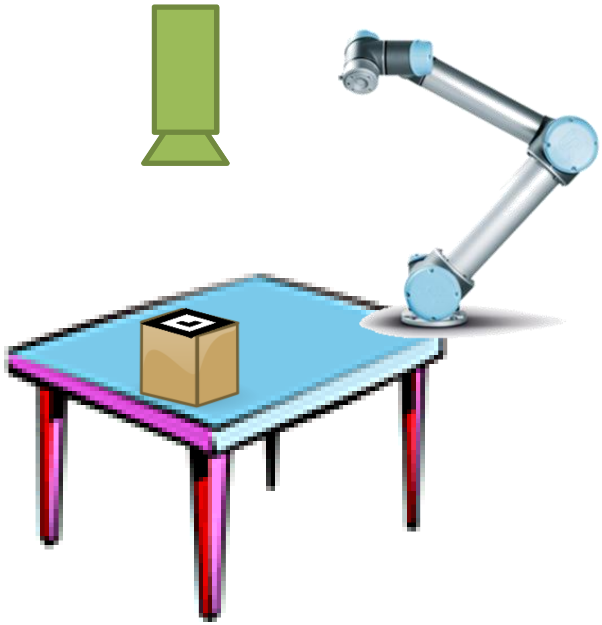
- **XACRO** is an XML-based “macro language” for building URDFs
  - <Include> other XACROs, with parameters
  - Simple expressions: math, substitution
- Used to build complex URDFs
  - multi-robot workcells
  - reuse standard URDFs (e.g. robots, tooling)

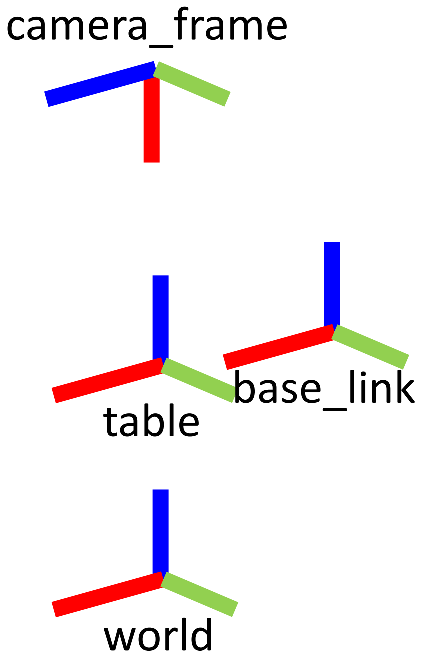
```
<xacro:include filename=“myRobot.xacro”/>

<xacro:myRobot prefix=“left_”/>
<xacro:myRobot prefix=“right_”/>

<property name=“offset” value=“1.3”/>

<joint name=“world_to_left" type=“fixed“>
    <parent link=“world"/>
    <child link=“left_base_link"/>
    <origin xyz=“${offset/2} 0 0" rpy="0 0 0"/>
</joint>
```

Exercise 3.1

Exercise 3.1

Combine simple urdf with ur5 xacro

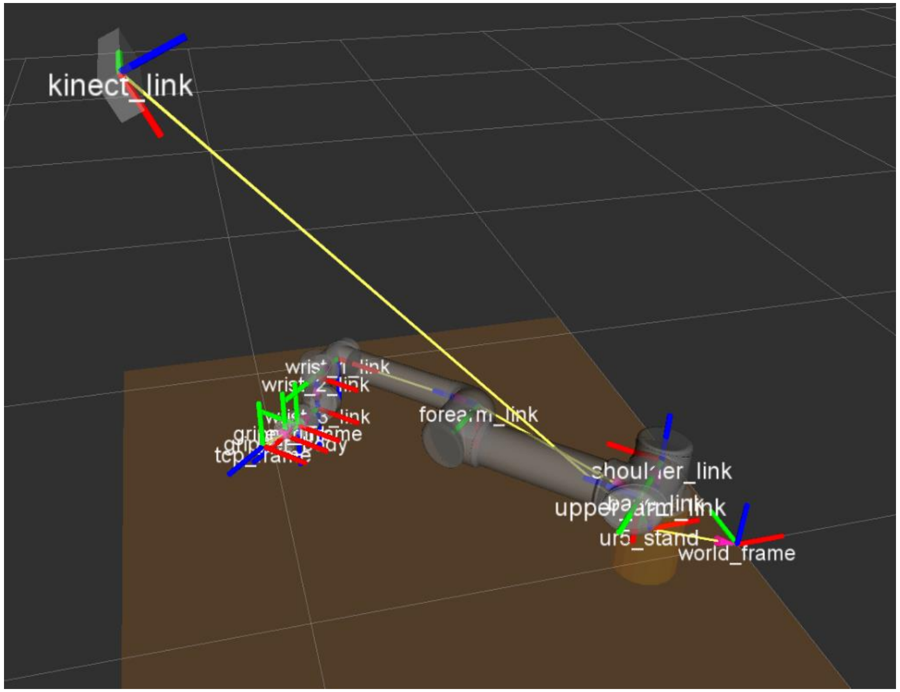




TF – Transforms in ROS

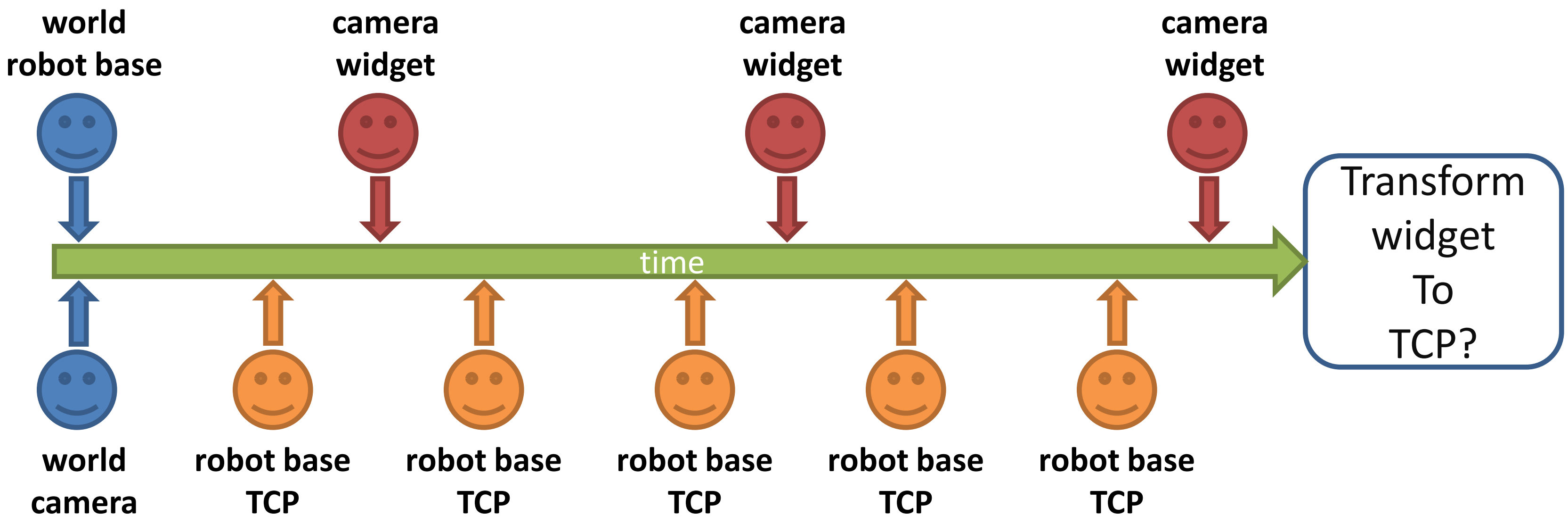
TF: Overview

- TF is a **distributed framework** to track **coordinate frames**
- Each frame is related to at least one other frame



TF: Time Sync

- TF tracks frame history
  - can be used to find transforms in the past!
  - essential for asynchronous / distributed system



TF: c++

- Each **node** has its own `transformListener`
  - listens to all tf messages, calculates relative transforms
  - Can try to transform in the past
  - ➢ Can only look as far back as it has been running

```
tf2_ros::Buffer buffer(node->get_clock());
tf2_ros::TransformListener listener(buffer);

geometry_msgs::msg::TransformStamped transform;
transform = buffer.lookupTransform("target", "source", tf2::TimePointZero);
```

Result

Parent Frame
("reference")

Child Frame
("object")

Time

- Note confusing "target/source" naming convention
- Tf2::TimePointZero gives **latest** available transform

TF Timing

- When requesting a transform, you must specify a **time**:
  - Latest Received

    ```
    lookupTransform("from", "to", tf2::TimePointZero)
    ```

  - Current Time (will probably fail)

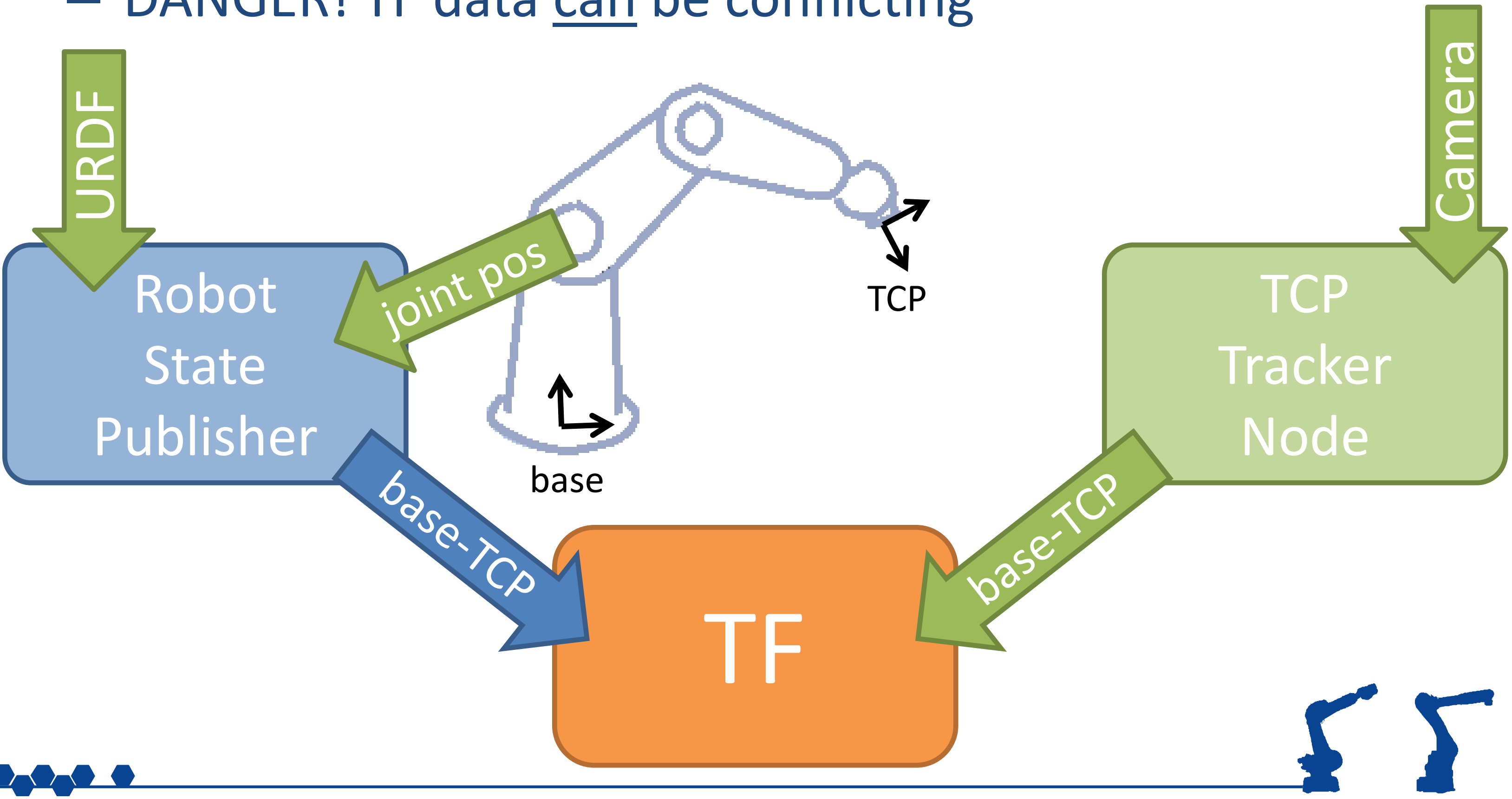
    ```
    lookupTransform("from", "to", now)
    ```

  - Current Time (wait for it to be available)

    ```
    lookupTransform("from", "to", now, 50ms)
    ```

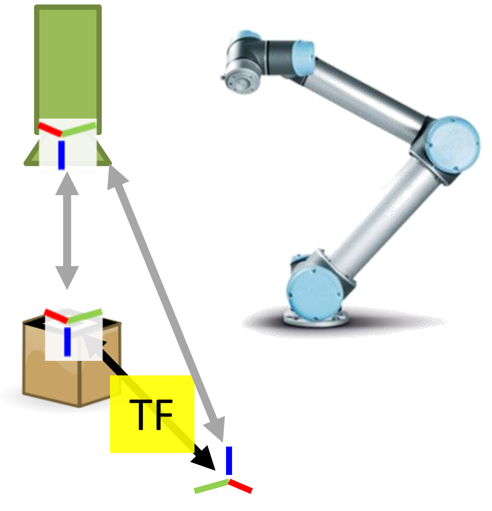
TF: Sources

- A robot_state_publisher provides TF data from a **URDF**
- Nodes can also publish TF data
  - DANGER! TF data can be conflicting



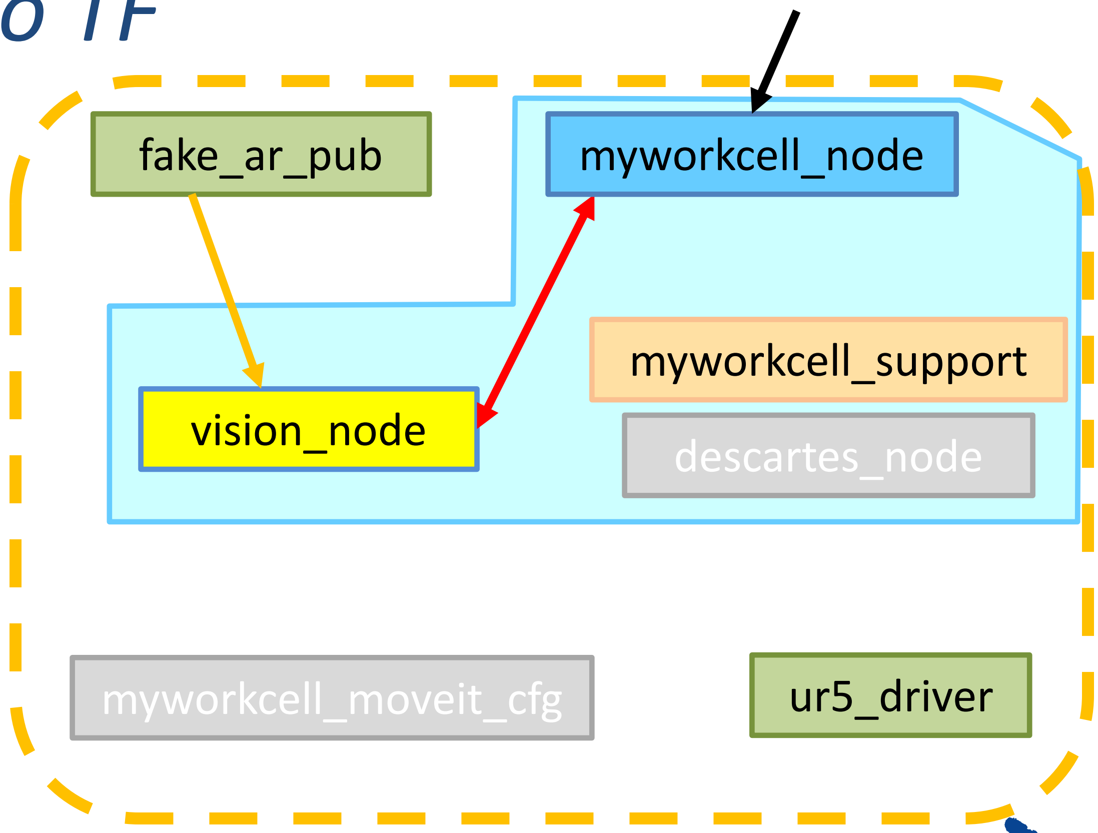
Exercise 3.2

Introduction to TF



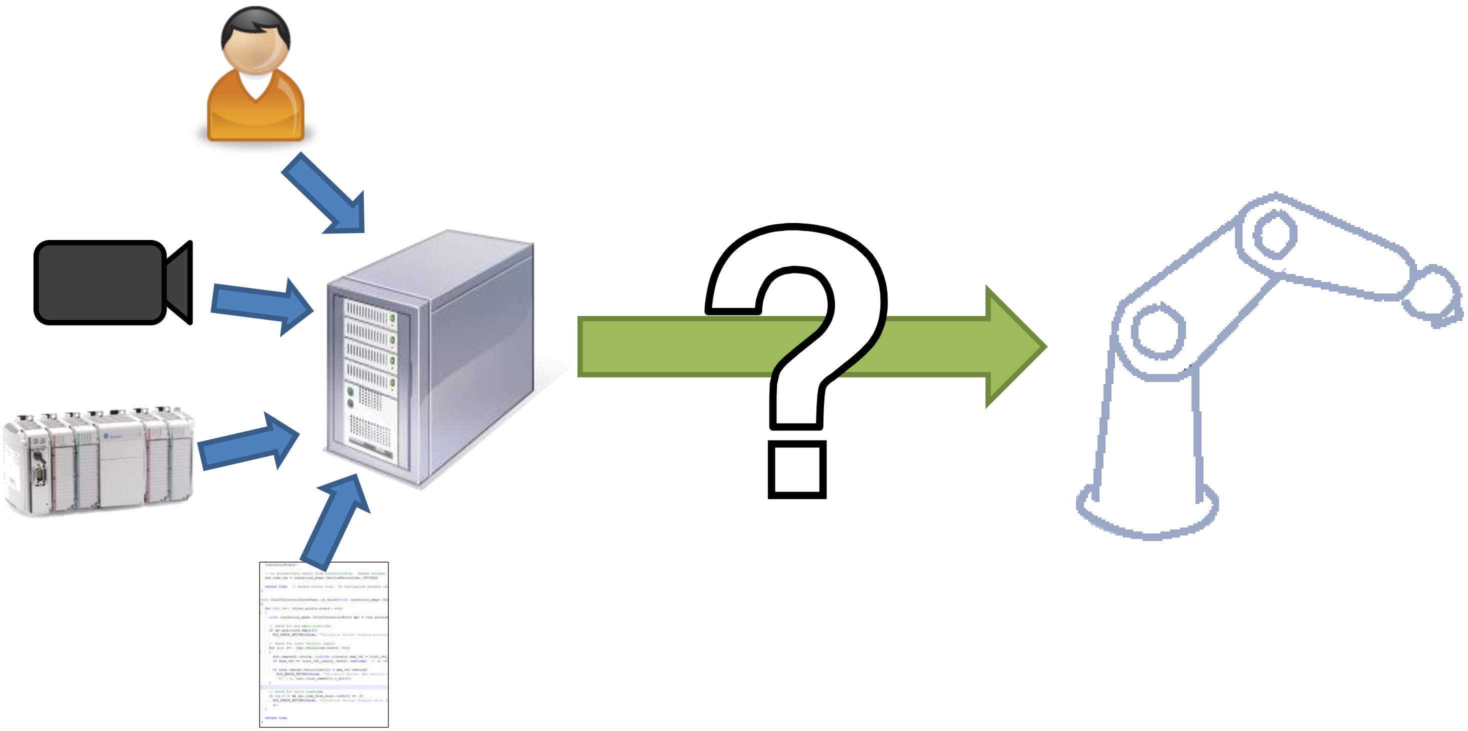
world->target = world->camera
* camera->target

Param: base_frame

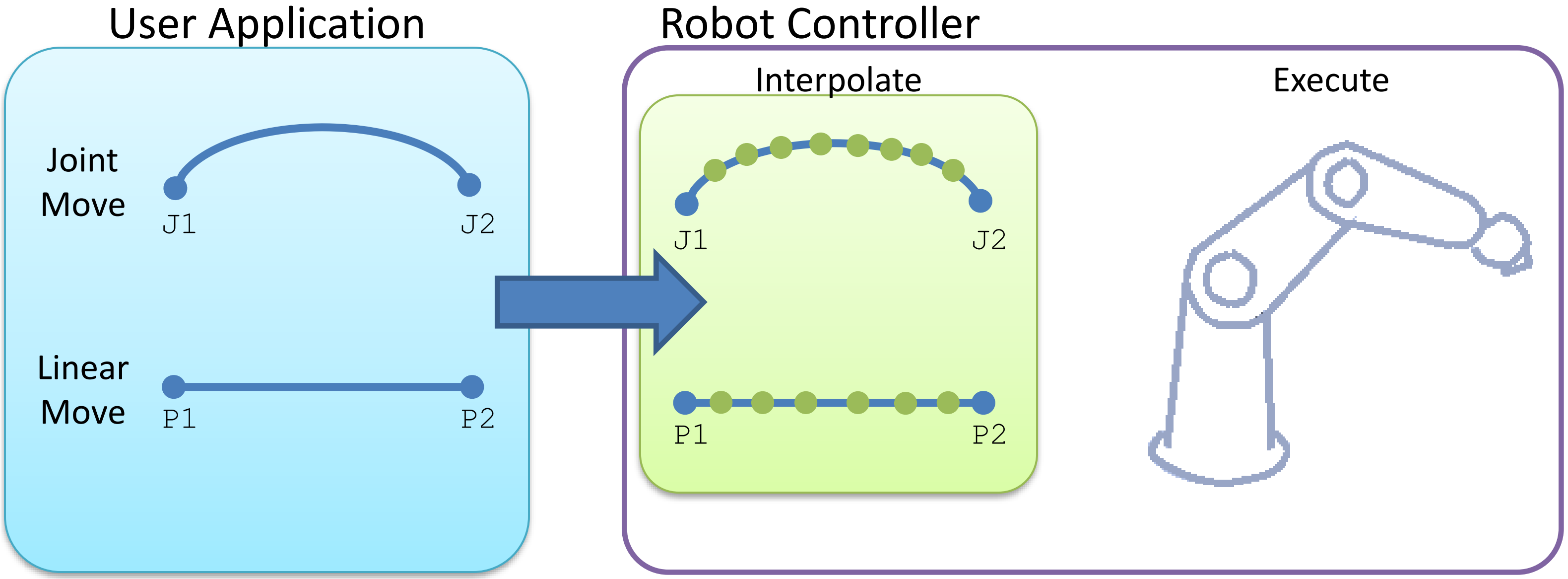


Motion Planning in ROS

Motion Planning in ROS

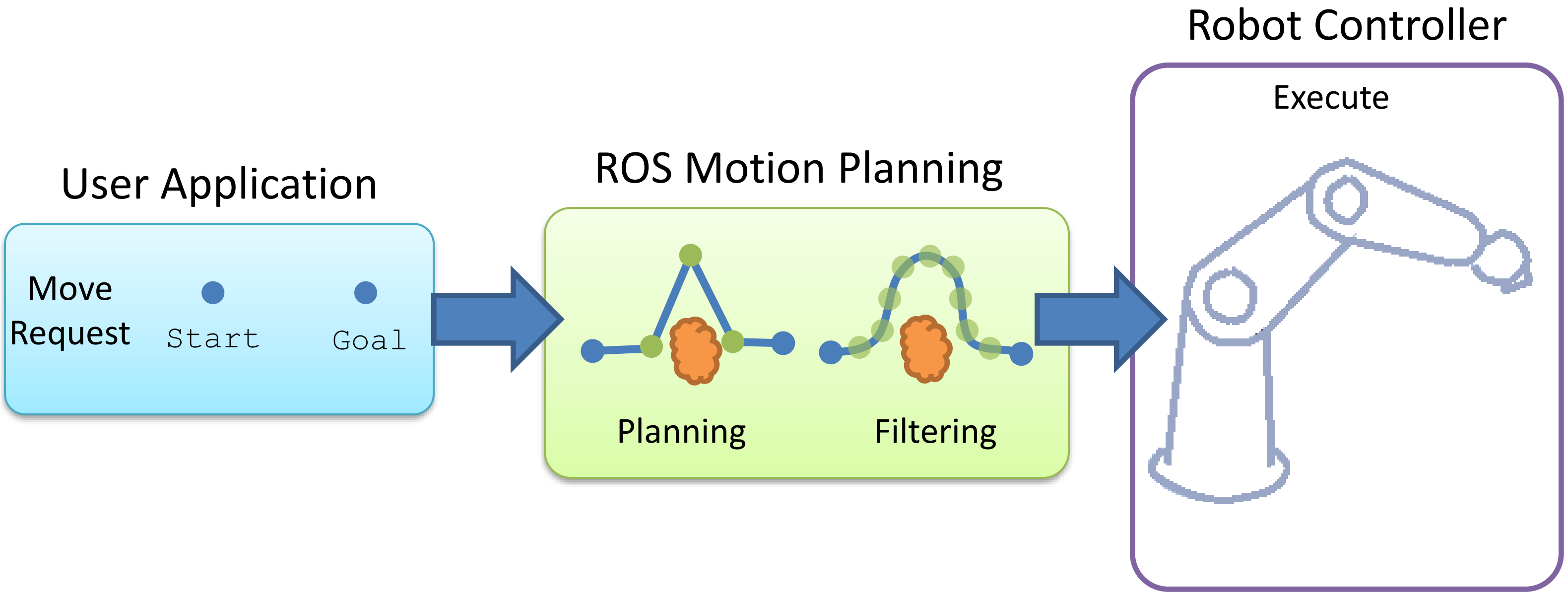


Traditional Robot Programming



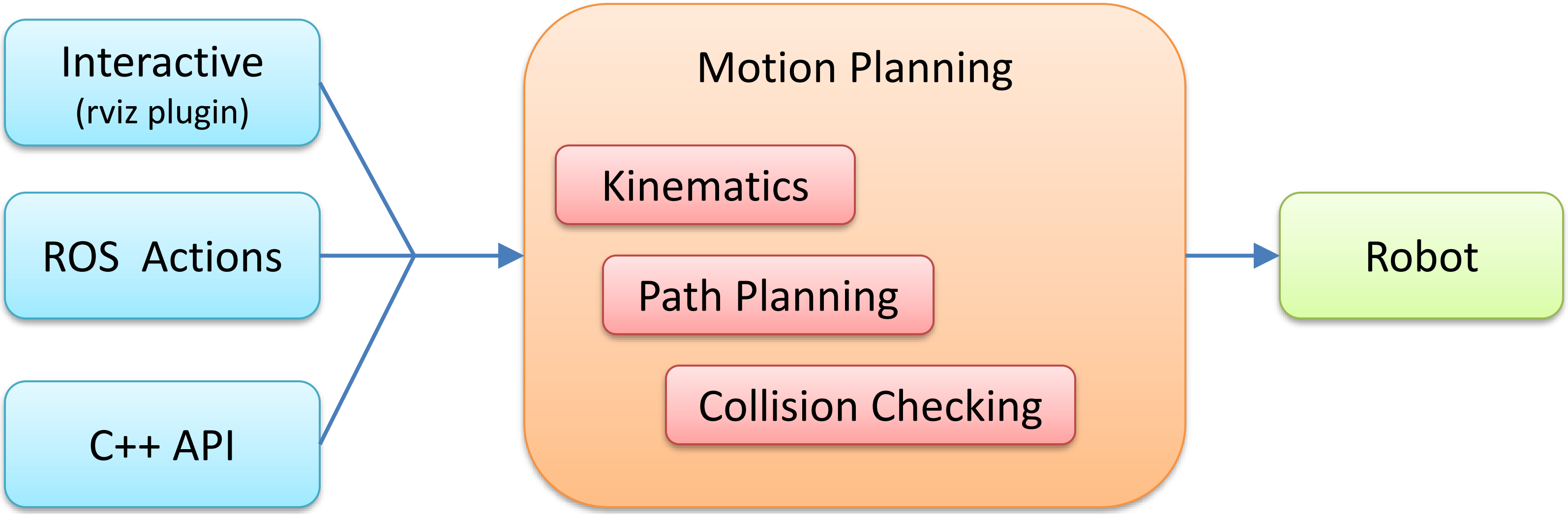
- Motion Types: *limited, but well-defined. One motion task.*
- Environment Model: *none*

ROS Motion Planning



- Motion Types: *flexible, goal-driven, with constraints*
  but minimal control over actual path
- Environment Model: *yes (fixed CAD or sensor-driven)*

Motion Planning Components



Interactive (rviz plugin)

ROS Actions

C++ API

Motion Planning

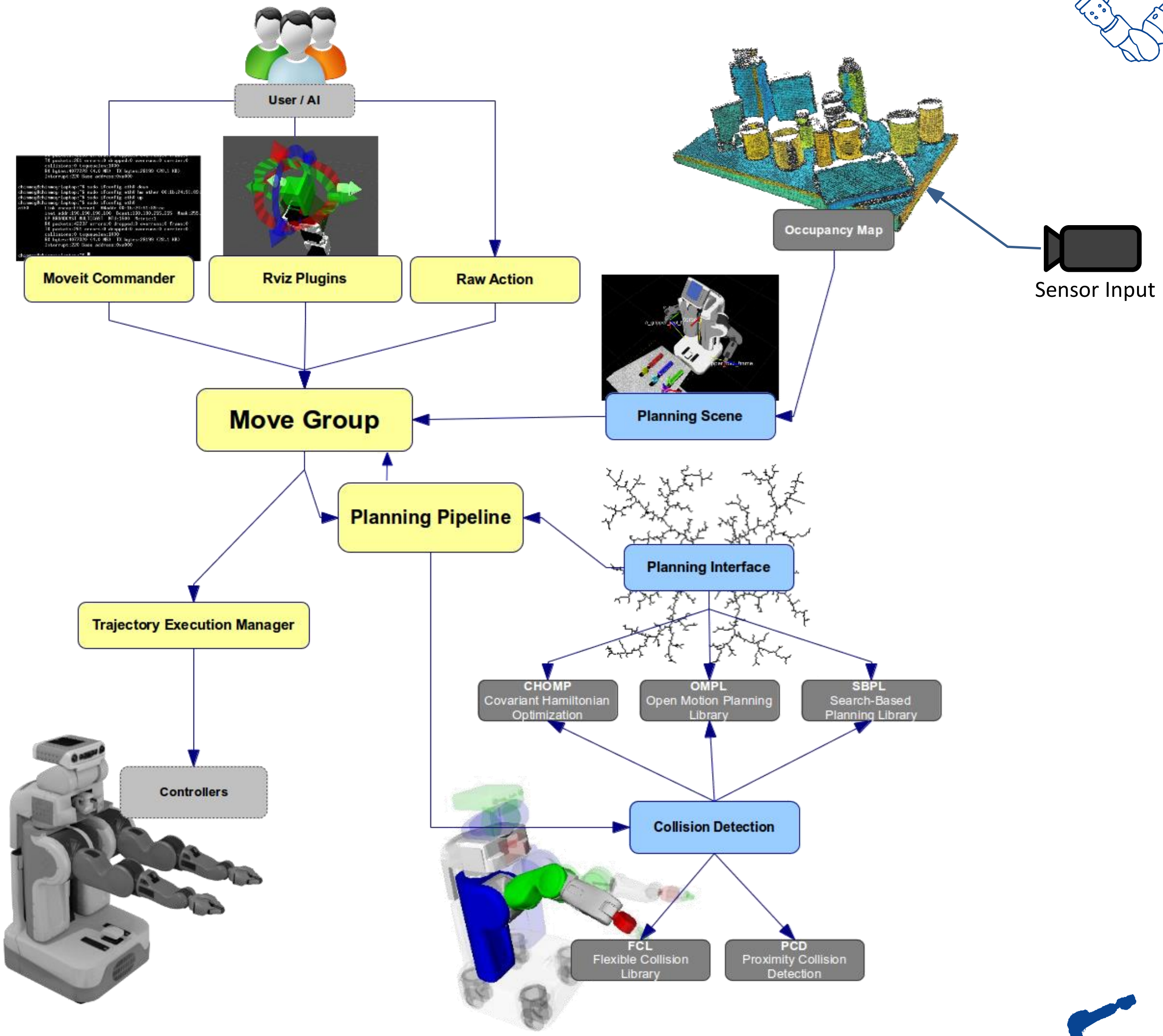
Kinematics

Path Planning

Collision Checking

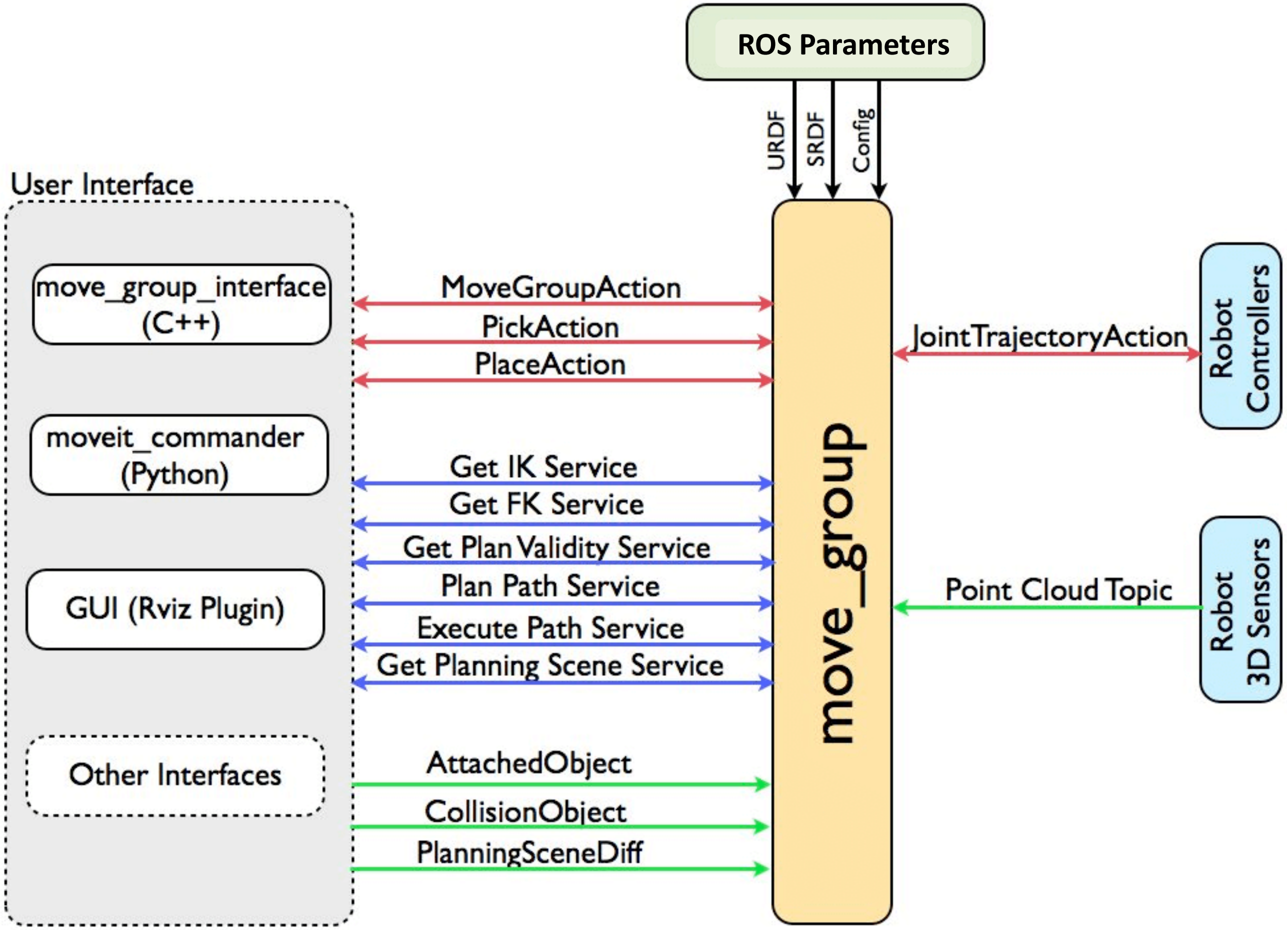
Robot

MoveIt Components



http://moveit.ros.org/wiki/High-level_Overview_Diagram
http://moveit.ros.org/wiki/Pipeline_Overview_Diagram

MoveIt Nodes



http://moveit.ros.org/documentation/concepts/

MoveIt! / Robot Integration

- A MoveIt! Package…
  - – includes all required nodes, config, launch files
    - motion planning, filtering, collision detection, etc.
  - – is unique to each individual robot model
    - includes references to URDF robot data
  - – uses a standard interface to robots
    - publish trajectory, listen to joint angles
  - – can (optionally) include workcell geometry
    - e.g. for collision checking

HowTo:

Set Up a New Robot

(or workcell)

Motivation

For each new robot model...

create a new MoveIt! package

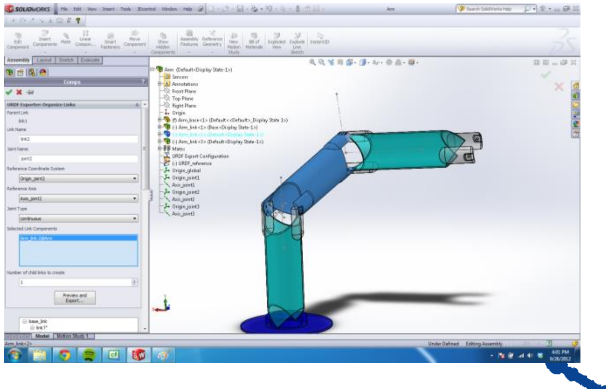
- Kinematics
  - physical configuration, lengths, etc.
- MoveIt! configuration
  - plugins, default parameter values
  - self-collision testing
  - pre-defined poses
- Robot connection
  - FollowJointTrajectory Action name

HowTo:
Set Up a New Robot

1. Create a URDF
2. Create a MoveIt! Package
3. Update MoveIt! Package for ROS-I
4. Test on ROS-I Simulator
5. Test on “Real” Robot

Create a URDF

- Previously covered URDF basics.
- Here are some tips:
  - create from datasheet or use [Solidworks Add-In]
  - double-check joint-offsets for accuracy
  - round near-zero offsets (if appropriate)
  - use "`base_link`" and "`tool0`"
  - use simplified collision models
    - convex-hull or primitives



URDF Exporter: Organize Links

Parent Link: link1

Link Name: link2

Joint Name: joint2

Reference Coordinate System: Origin_joint2

Reference Axis: Axis_joint2

Joint Type: continuous

Selected Link Components: Arm_link-2@Arm

Number of child links to create: 1

base_link
link1*

Arm_link<2>

Verify the URDF

- It is **critical** to verify that your URDF matches the physical robot:
  - each joint moves as expected
  - joint-coupling issues are identified
  - min/max joint limits
  - joint directions (pos/neg)
  - correct zero-position, etc.
  - check forward kinematics

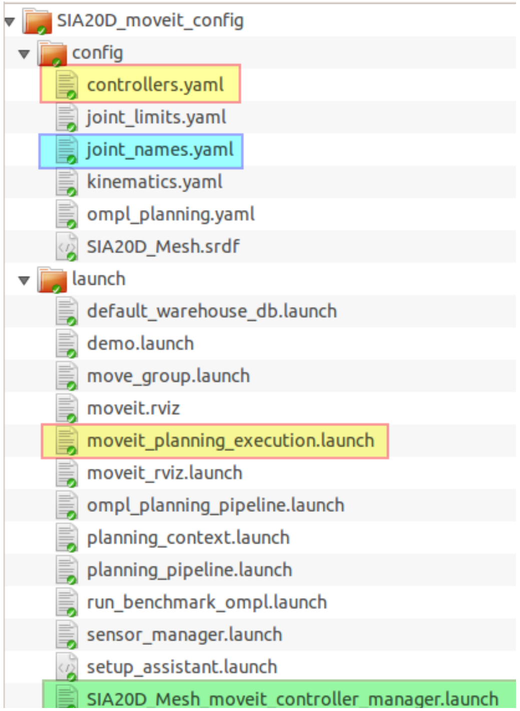
Create a MoveIt! Package

- Use the MoveIt! Setup Assistant
  - can create a new package or edit an existing one

Coming Soon to ROS2!

Update MoveIt! Package

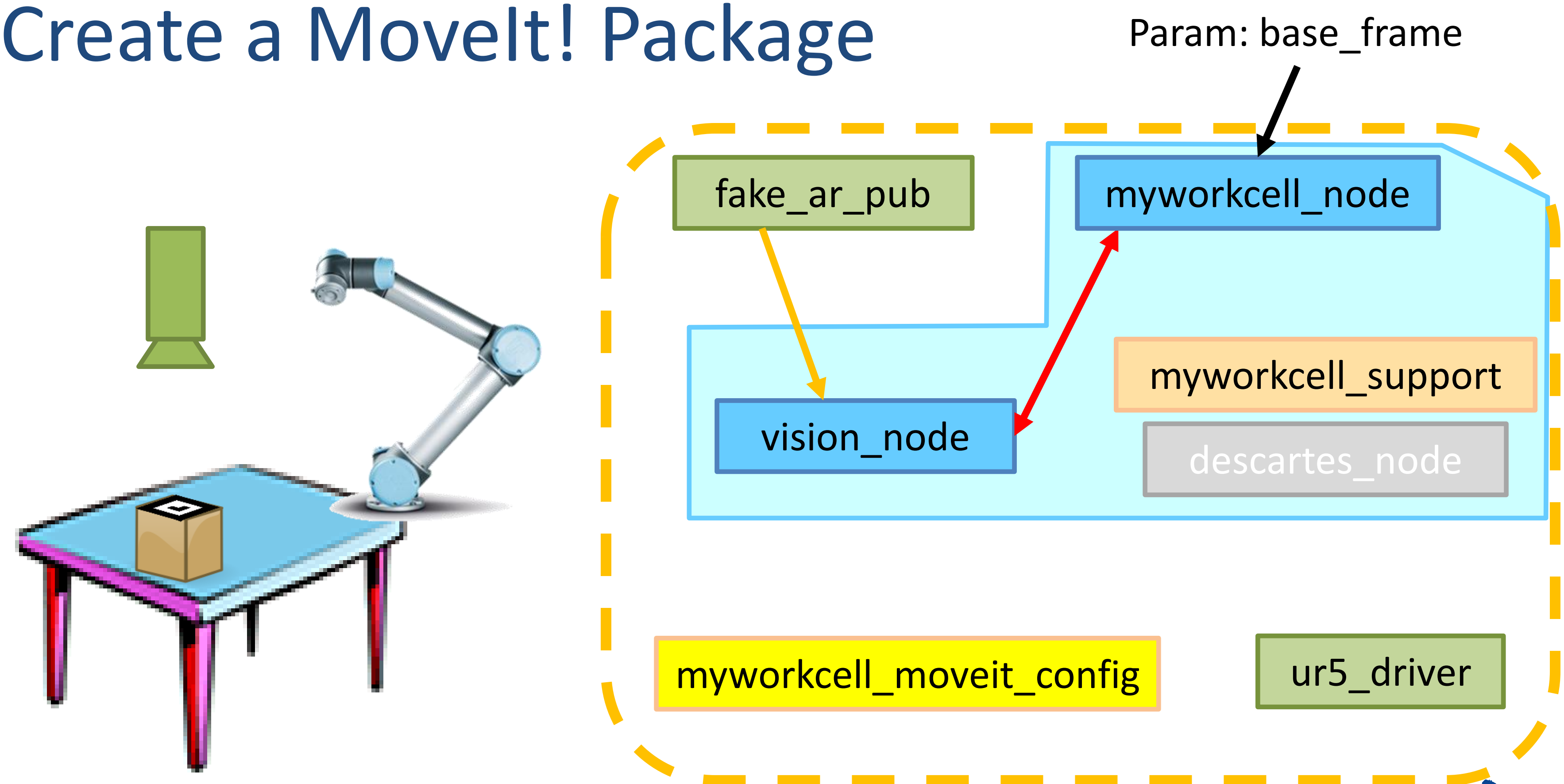
- Setup Assistant generates a *generic* package
  - missing config. data to connect to a specific robot
  - ROS-I robots use a *standard* interface



- SIA20D_moveit_config
  - config
    - controllers.yaml
    - joint_limits.yaml
    - joint_names.yaml
    - kinematics.yaml
    - ompl_planning.yaml
    - SIA20D_Mesh.srdf
  - launch
    - default_warehouse_db.launch
    - demo.launch
    - move_group.launch
    - moveit.rviz
    - moveit_planning_execution.launch
    - moveit_rviz.launch
    - ompl_planning_pipeline.launch
    - planning_context.launch
    - planning_pipeline.launch
    - run_benchmark_ompl.launch
    - sensor_manager.launch
    - setup_assistant.launch
    - SIA20D_Mesh_moveit_controller_manager.launch

Exercise 3.3:

Create a MoveIt! Package

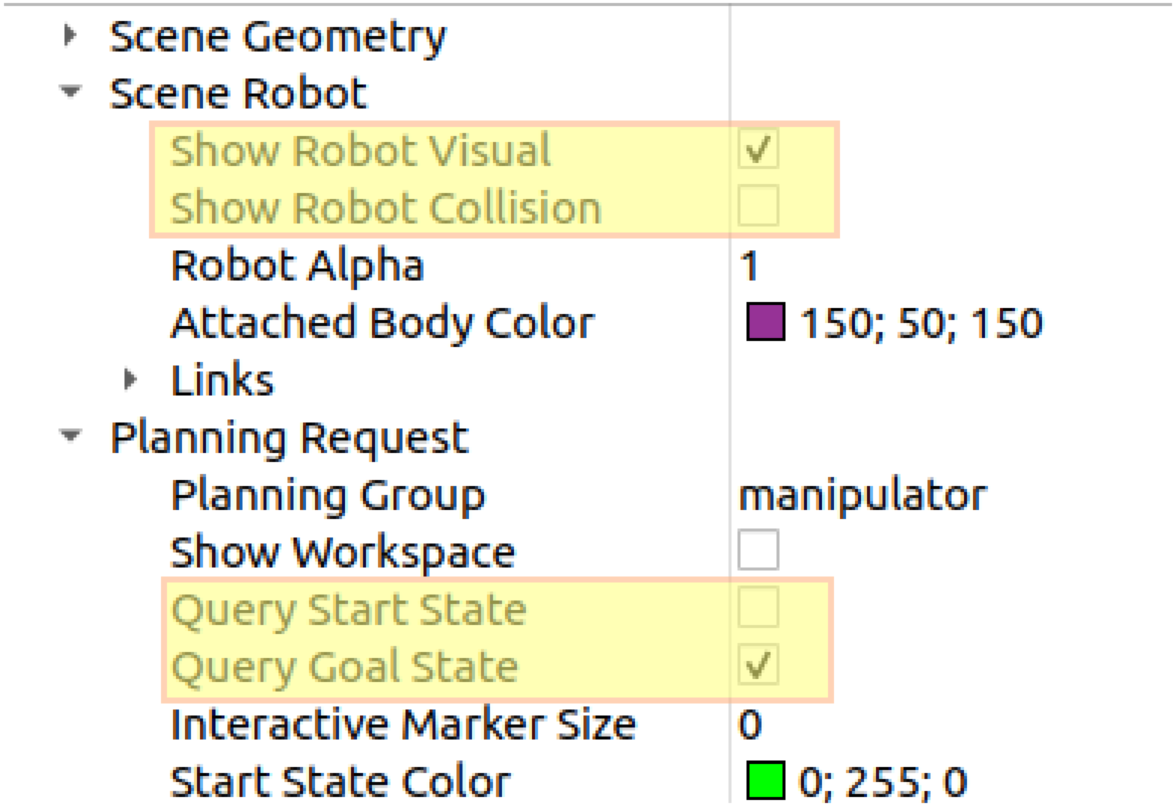


HowTo:
Motion Planning using MoveIt!

1. Motion Planning using Rviz
2. Motion Planning using C++

Motion Planning in RViz

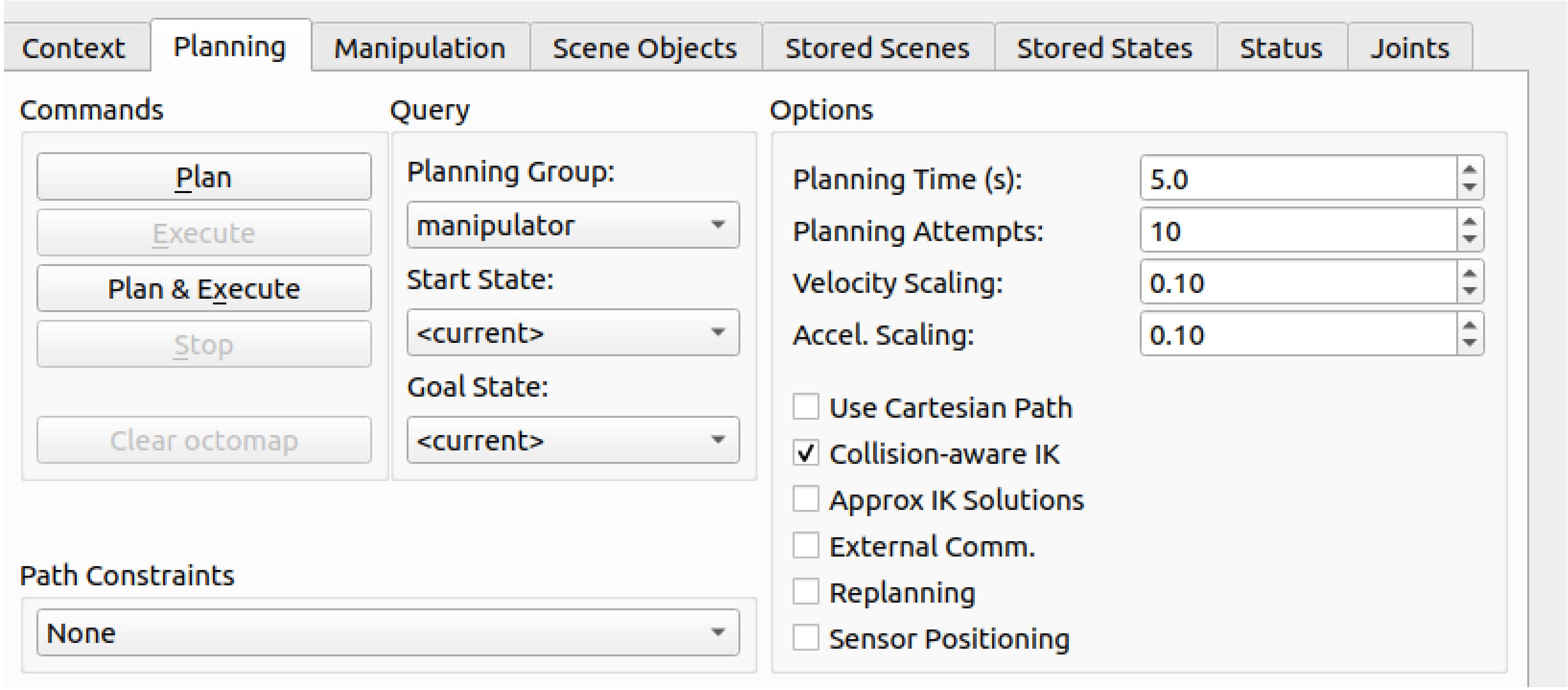
Display Options



▸ Scene Geometry	
▾ Scene Robot	
Show Robot Visual	☑
Show Robot Collision	☐
Robot Alpha	1
Attached Body Color	🟪 150; 50; 150
▸ Links	
▾ Planning Request	
Planning Group	manipulator
Show Workspace	☐
Query Start State	☐
Query Goal State	☑
Interactive Marker Size	0
Start State Color	🟩 0; 255; 0

Motion Planning in RViz

Planning Options



Query

Planning Group: manipulator

Start State: <current>

Goal State: <current>

Path Constraints: None

Options

Planning Time (s): 5.0

Planning Attempts: 10

Velocity Scaling: 0.10

Accel. Scaling: 0.10

☐ Use Cartesian Path

☑ Collision-aware IK

☐ Approx IK Solutions

☐ External Comm.

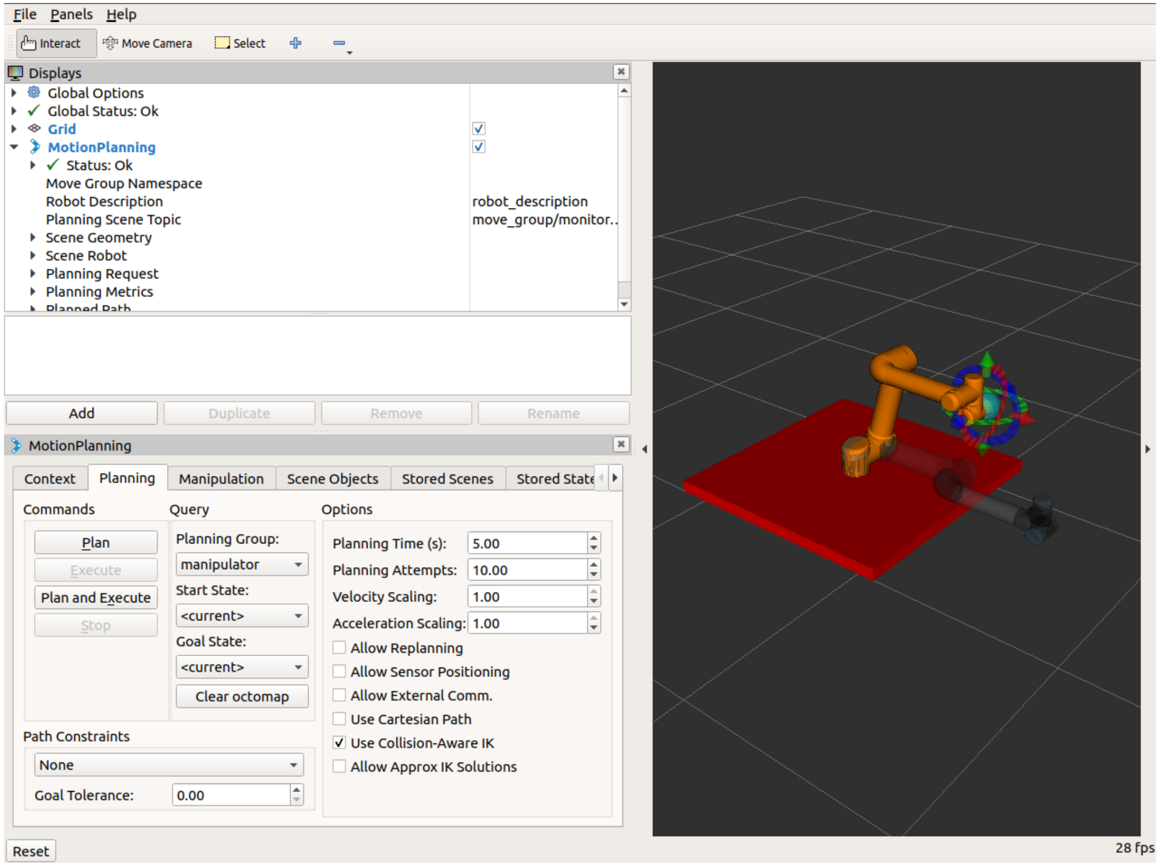
☐ Replanning

☐ Sensor Positioning

Exercise 3.4

Exercise 3.4:

Motion Planning using RVIZ



Global Options
Global Status: Ok
Grid ☑
MotionPlanning ☑
Status: Ok
Move Group Namespace
Robot Description: robot_description
Planning Scene Topic: move_group/monitor..
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path

Query
Planning Group: manipulator
Start State: <current>
Goal State: <current>

Options
Planning Time (s): 5.00
Planning Attempts: 10.00
Velocity Scaling: 1.00
Acceleration Scaling: 1.00
☐ Allow Replanning
☐ Allow Sensor Positioning
☐ Allow External Comm.
☐ Use Cartesian Path
☑ Use Collision-Aware IK
☐ Allow Approx IK Solutions

Path Constraints: None
Goal Tolerance: 0.00

Review

ROS

- URDF
- MoveIt
- Path Planners
- RViz Planning

ROS-Industrial

- Robot Drivers
- Path Planners

Questions?

- ROS-I Architecture
- Setup Assistant
- Robot Launch Files
- RViz Planning