ROS-Industrial Basic Developer's Training Class

February 2017

Southwest Research Institute

Session 3:
Motion Control of Manipulators

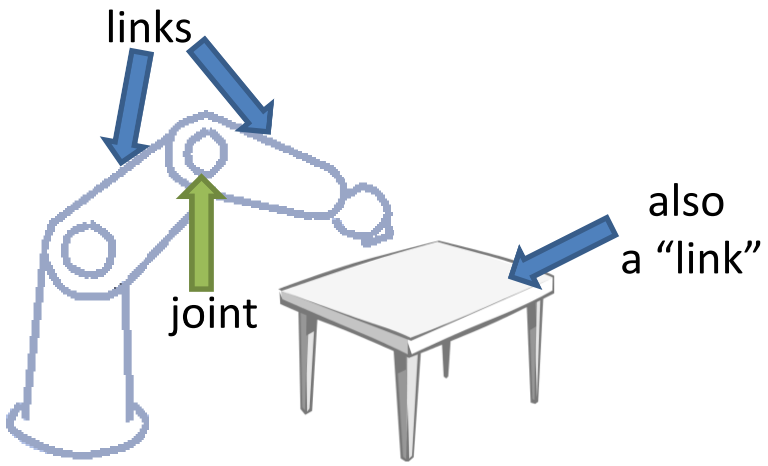
February 2017

Southwest Research Institute

URDF: Unified Robot Description Format

URDF: Overview

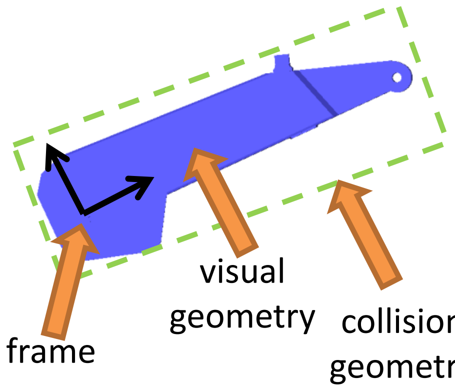
- URDF is an XML-formatted file containing:
  - **Links**: coordinate frames and associated geometry
  - **Joints:** connections between links
- Similar to DH-parameters (but much less difficult)
- Can describe entire workspace, not just robots



URDF: Link

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

```
<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>
```

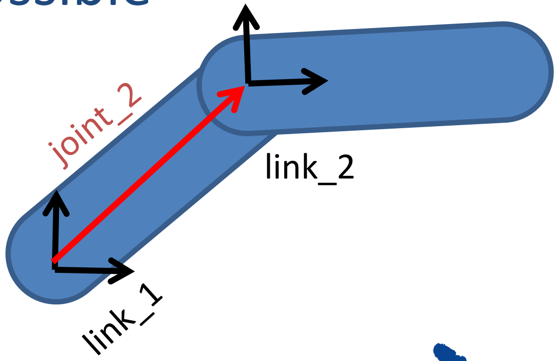


URDF Transforms	
X/Y/Z	Roll/Pitch/Yaw
Meters	Radians

URDF: Joint

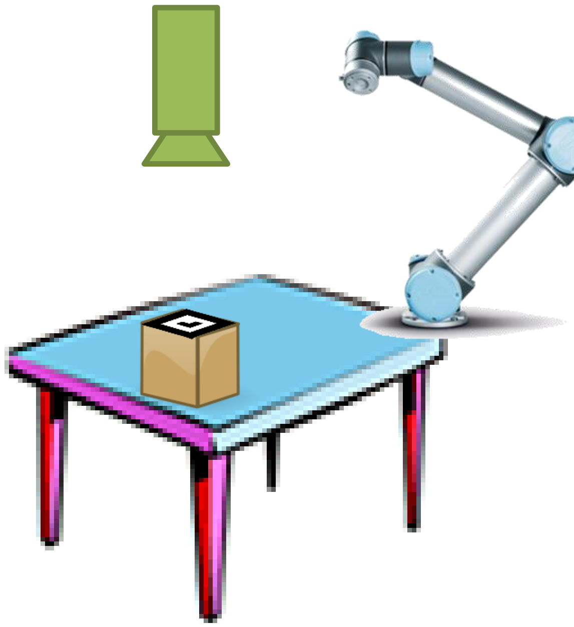
- A **Joint** connects two **Links**
  - Defines a **transform** between **parent** and **child** frames
    - Types: *fixed, free, linear, rotary*
  - Denotes axis of movement *(for linear / rotary)*
  - Contains joint limits on position and velocity
- ROS-I conventions
  - X-axis front, Z-Axis up
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

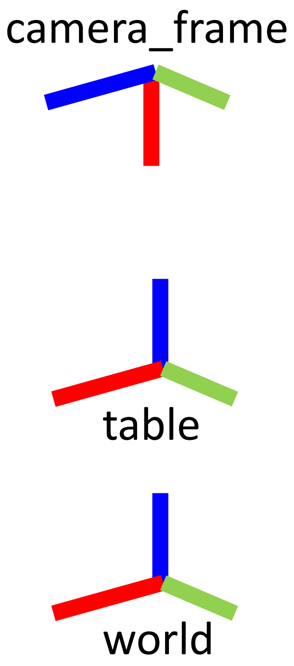
```
<joint name="joint_2" type="revolute">
   <parent link="link_1"/>
   <child link="link_2"/>
   <origin xyz="0.2 0.2 0" rpy="0 0 0"/>
   <axis xyz="0 0 1"/>
   <limit lower="-3.14" upper="3.14" velocity="1.0"/>
</joint>
```



Exercise 3.0

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)

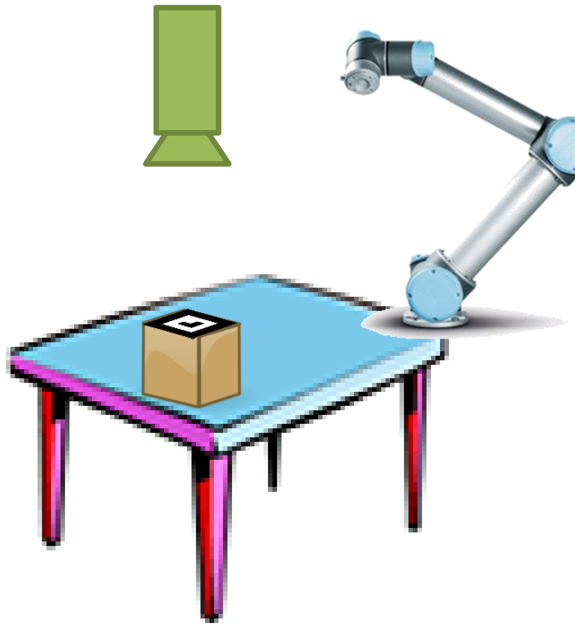
```
<xacro:include filename=“myRobot.xacro”/>

<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>
```

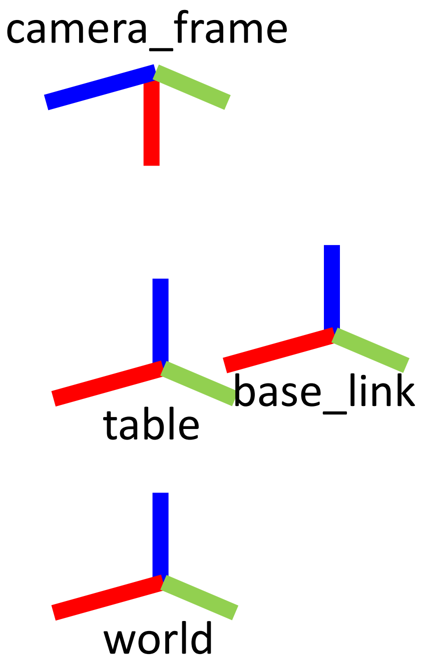
URDF Practical Examples

- Let’s take a quick look at the UR5’s URDF:
  - *In **ur_description/urdf/ur5.urdf.xacro***

Exercise 3.1

Combine simple urdf with ur5 xacro

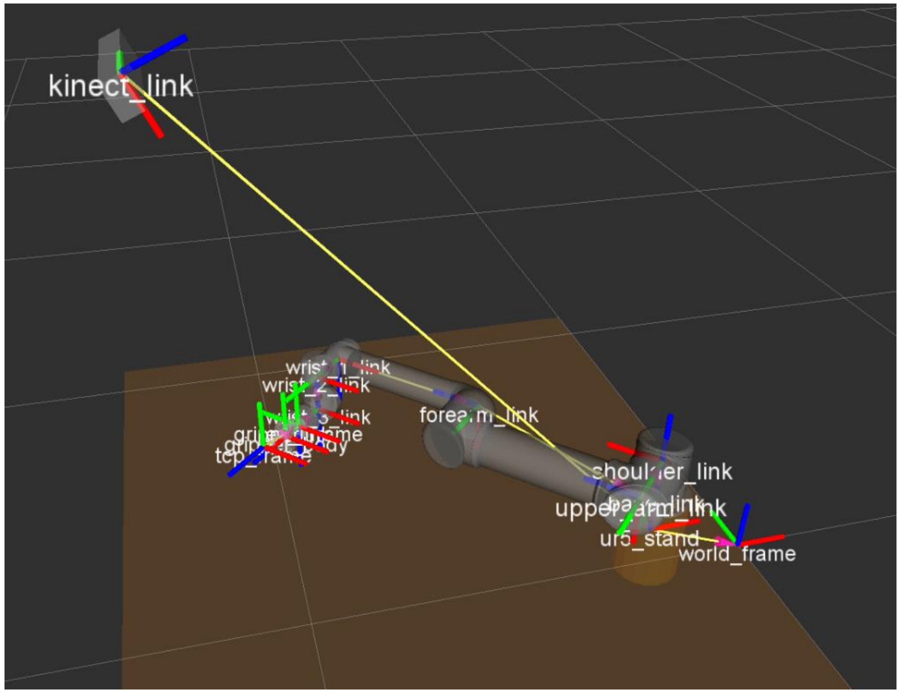




TF – Transforms in ROS

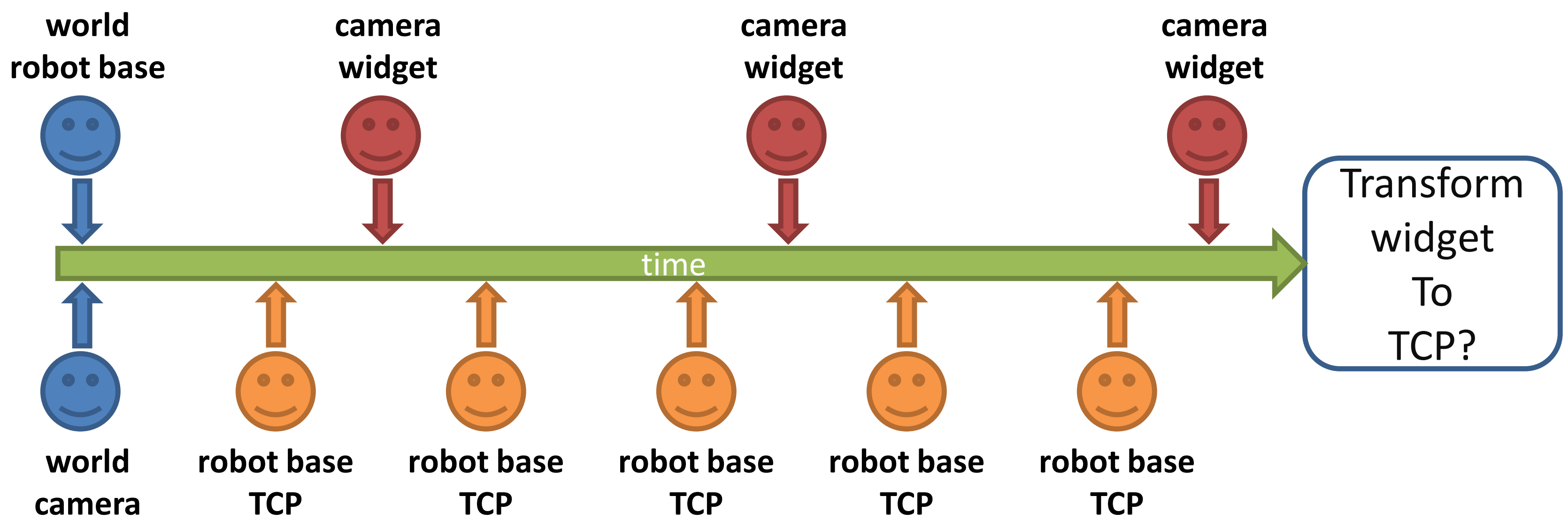
TF: Overview

- 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

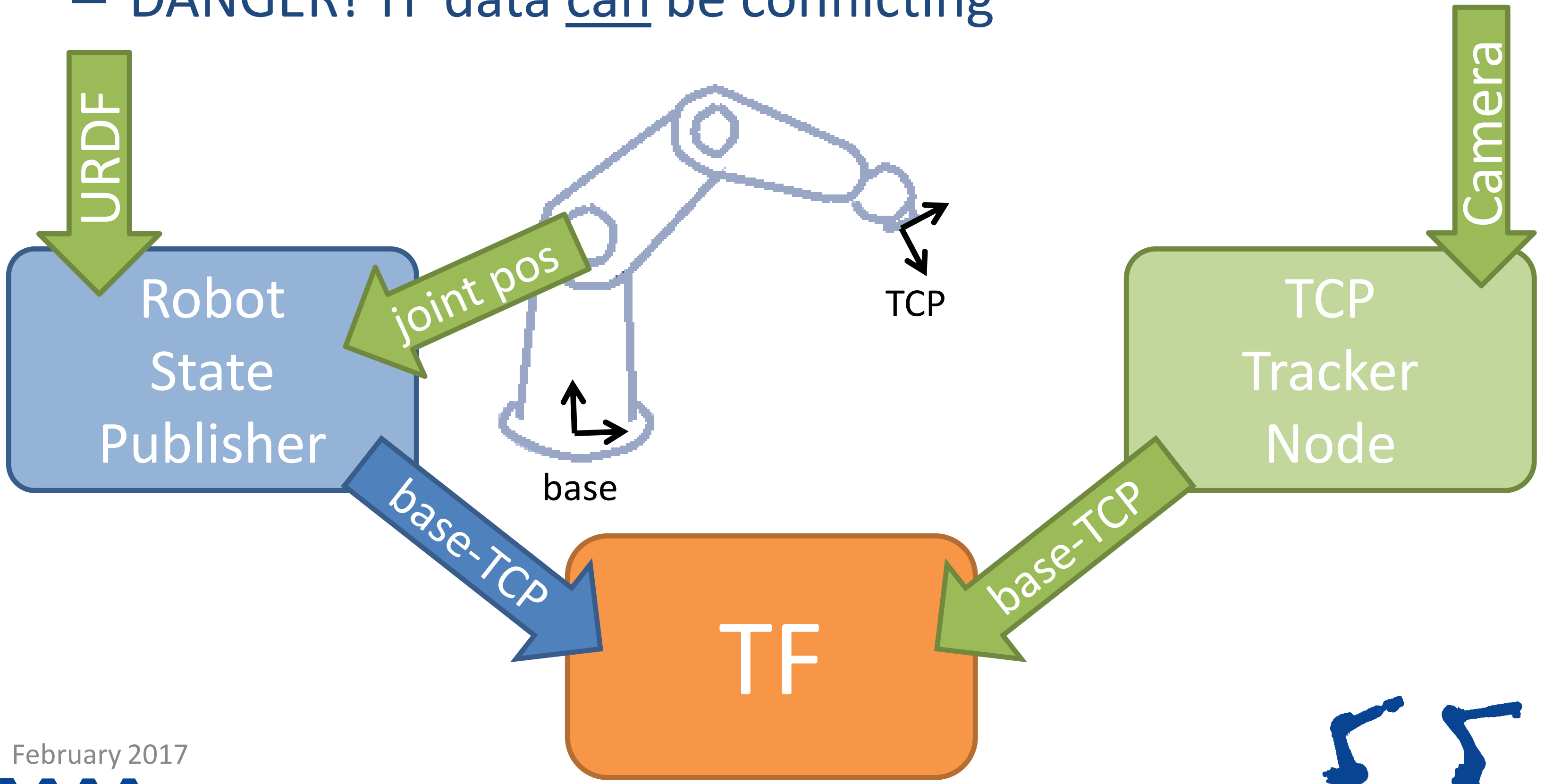
```
tf::TransformListener listener;
tf::StampedTransform transform;

listener.lookupTransform("target", "source", ros::Time(), transform);
```

Parent Frame ("reference")　Child Frame ("object")　Time　Result

- Note confusing "target/source" naming convention
- ros::Time() or ros::Time(0) give **latest** available transform
- ros::Time::now() usually fails

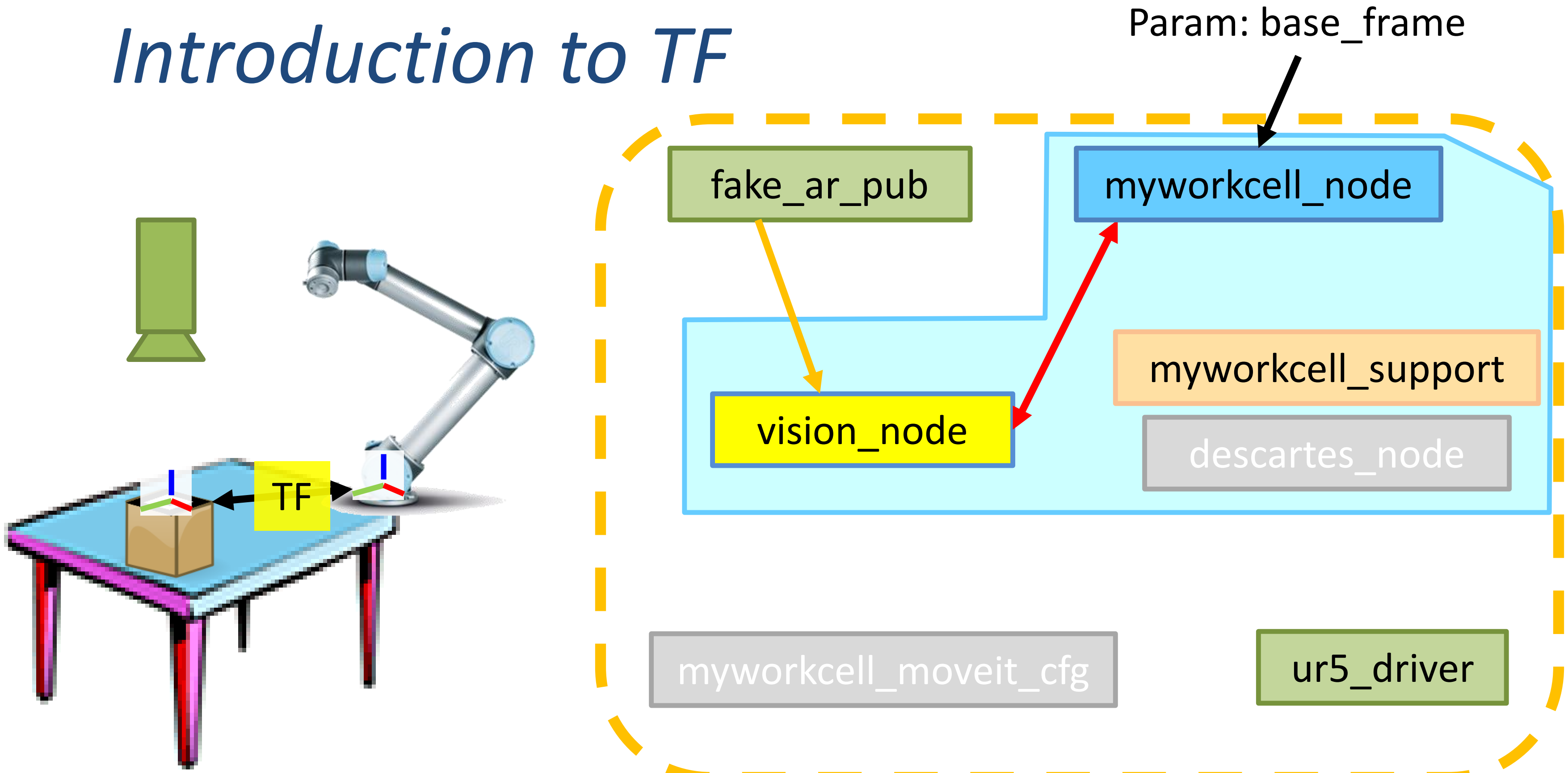
TF: Sources

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



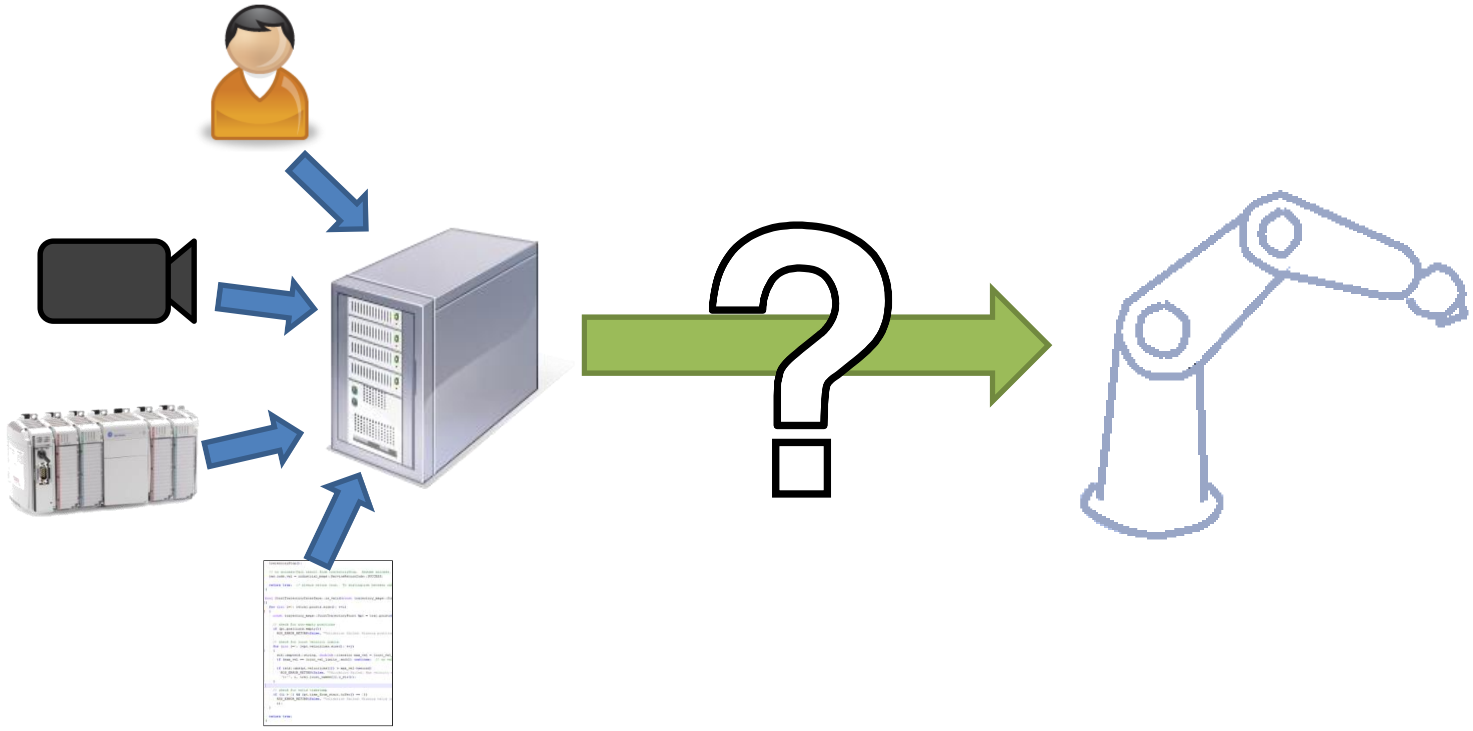
Exercise 3.2

Introduction to TF

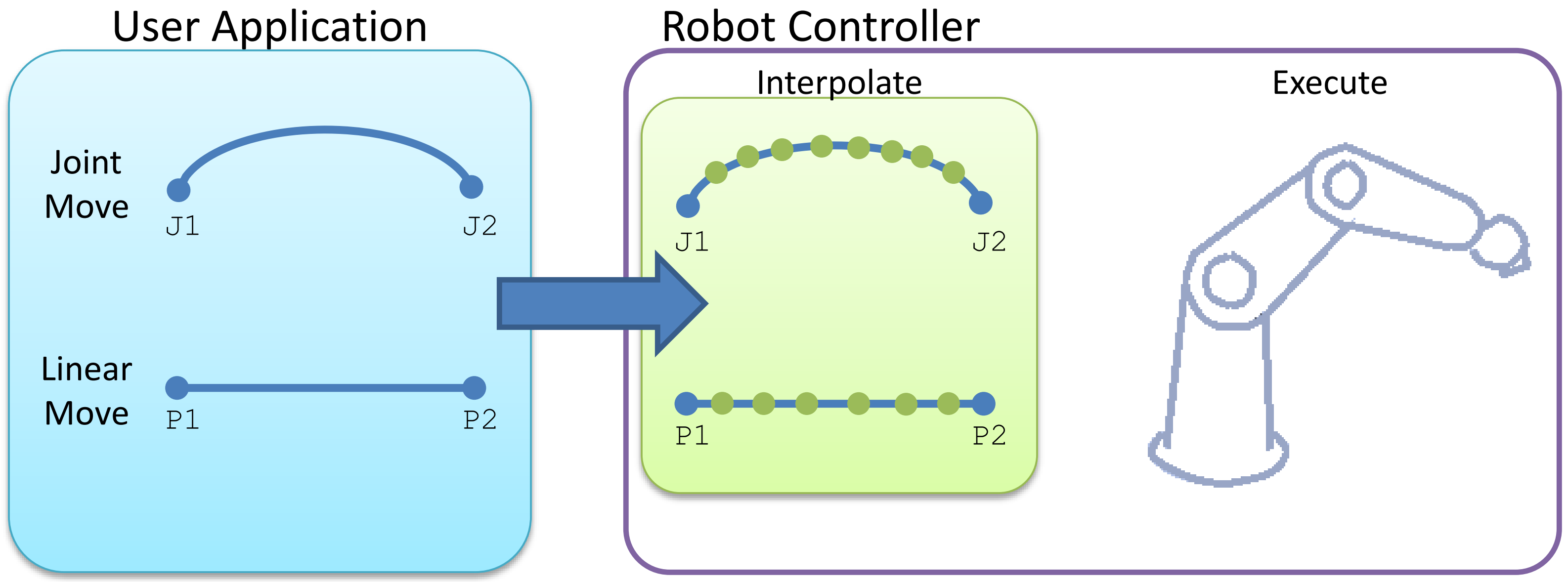


Motion Planning in ROS

Motion Planning in ROS

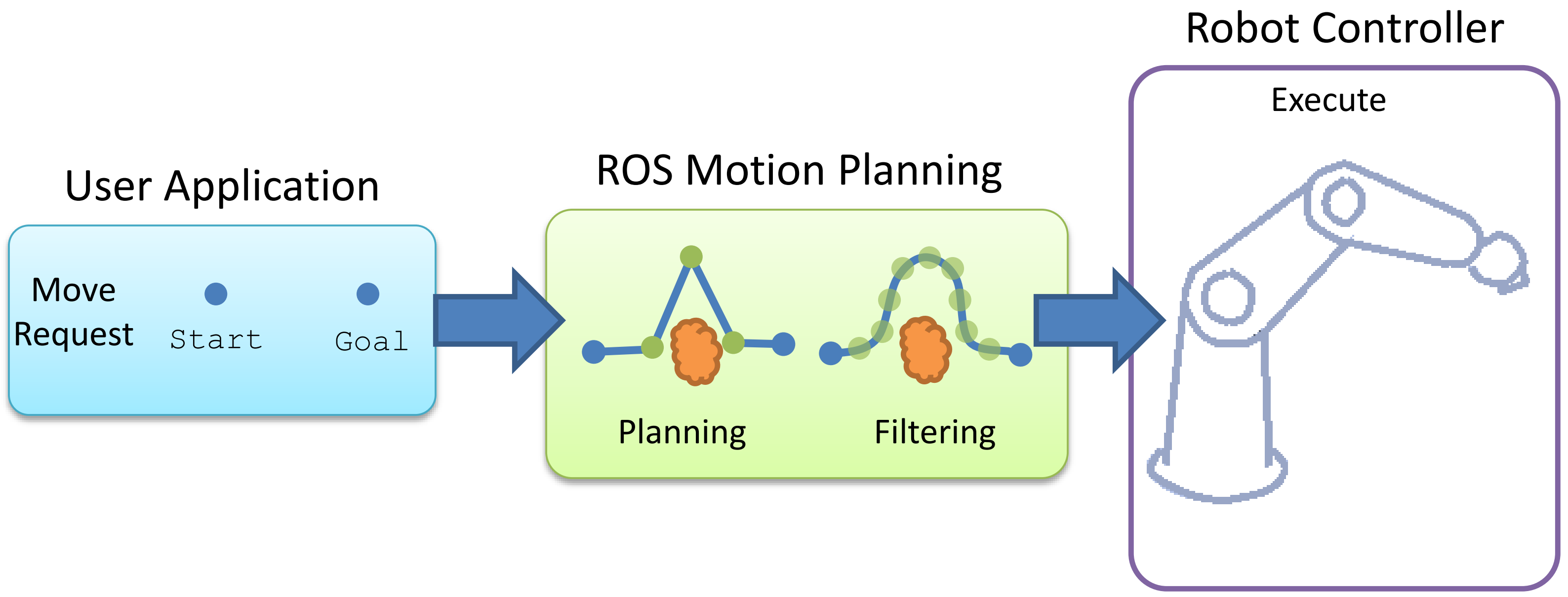


Traditional Robot Programming



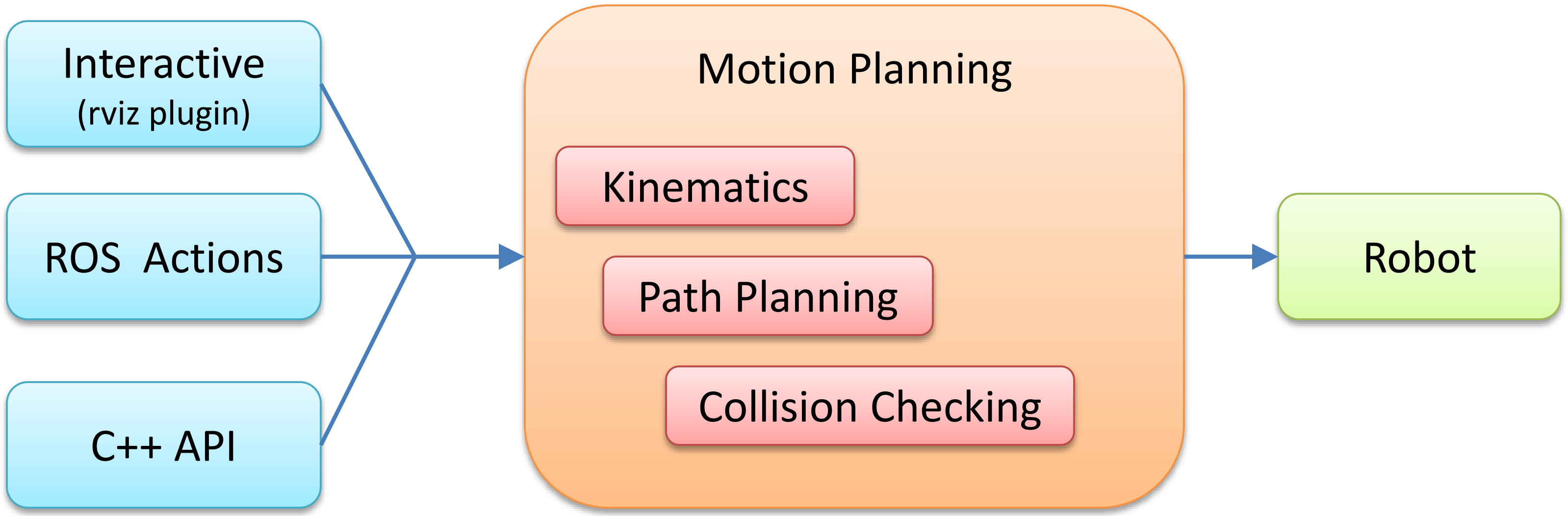
- Motion Types: *limited, but well-defined. One motion task.*
- Environment Model: *none*
- Execution Monitor: *application-specific*

ROS Motion Planning

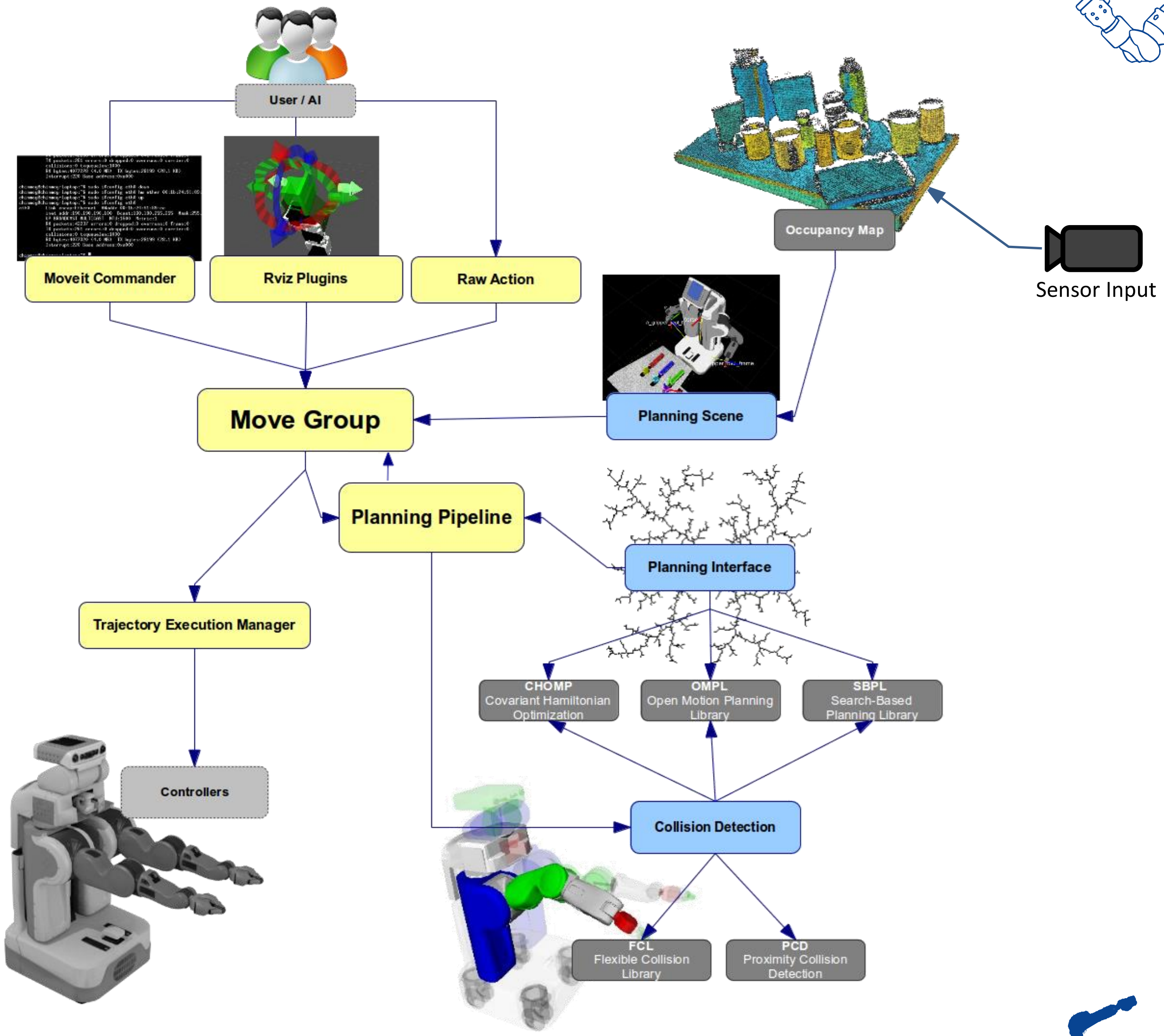


- Motion Types: *Flexible, goal-driven, with constraints*
  but minimal control over actual path
- Environment Model: *Automatic, based on live sensor feedback*
- Execution Monitor: *Detects changes during motion*

Motion Planning Components

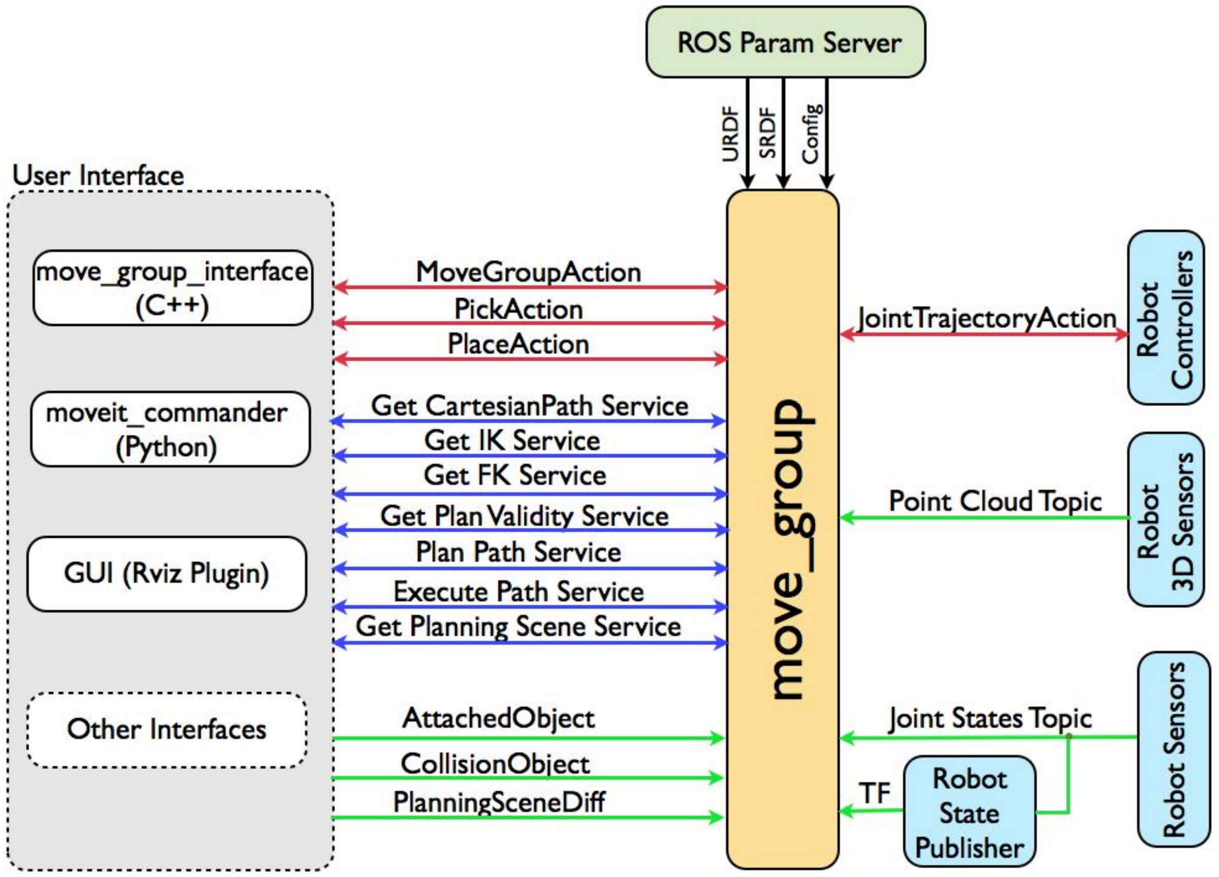


MoveIt Components



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

- 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

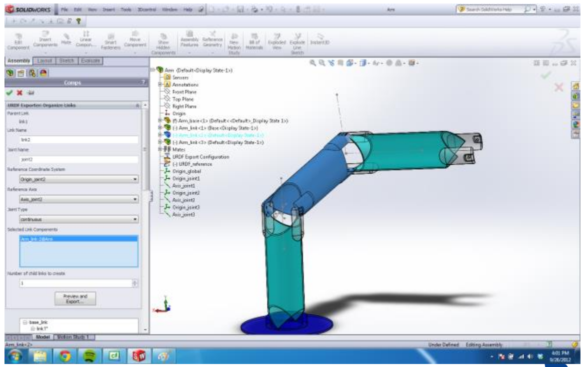
- Kinematics
  - Physical configuration, lengths, etc.
- MoveIt! configuration
  - Plugins, default parameter values
  - Self-collision testing
  - Pre-defined poses
- Robot connection
  - FollowJointTrajectory Action name

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



Verify the URDF

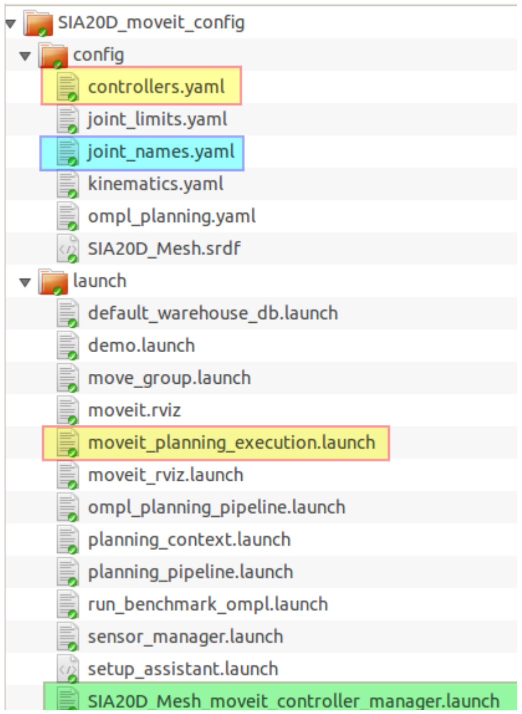
- It is **critical** to verify that your URDF matches the physical robot
  - Each joint moves as expected
  - Joint-coupling issues are identified
  - Min/max joint limits
  - Joint directions (pos/neg)
  - Correct zero-position, etc.
  - Check forward kinematics

Create a MoveIt! Package

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

Update MoveIt! Package

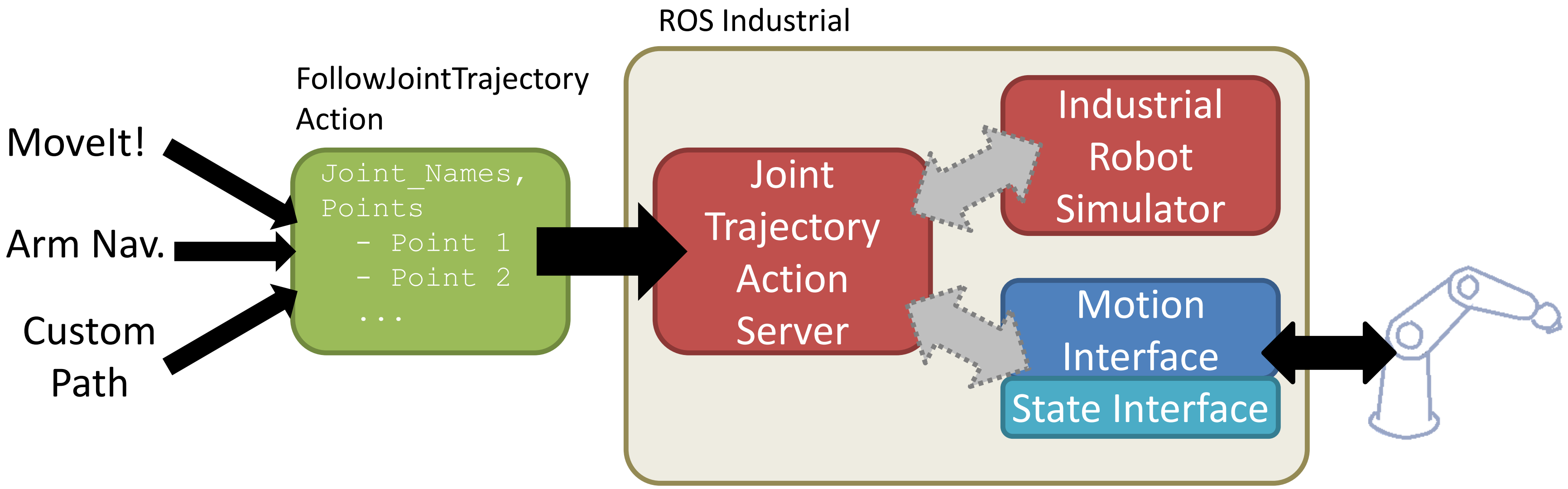
- Setup Assistant generates a *generic* package
  - Missing config. data to connect to a specific robot
  - ROS-I robots use a *standard* interface



```
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
```

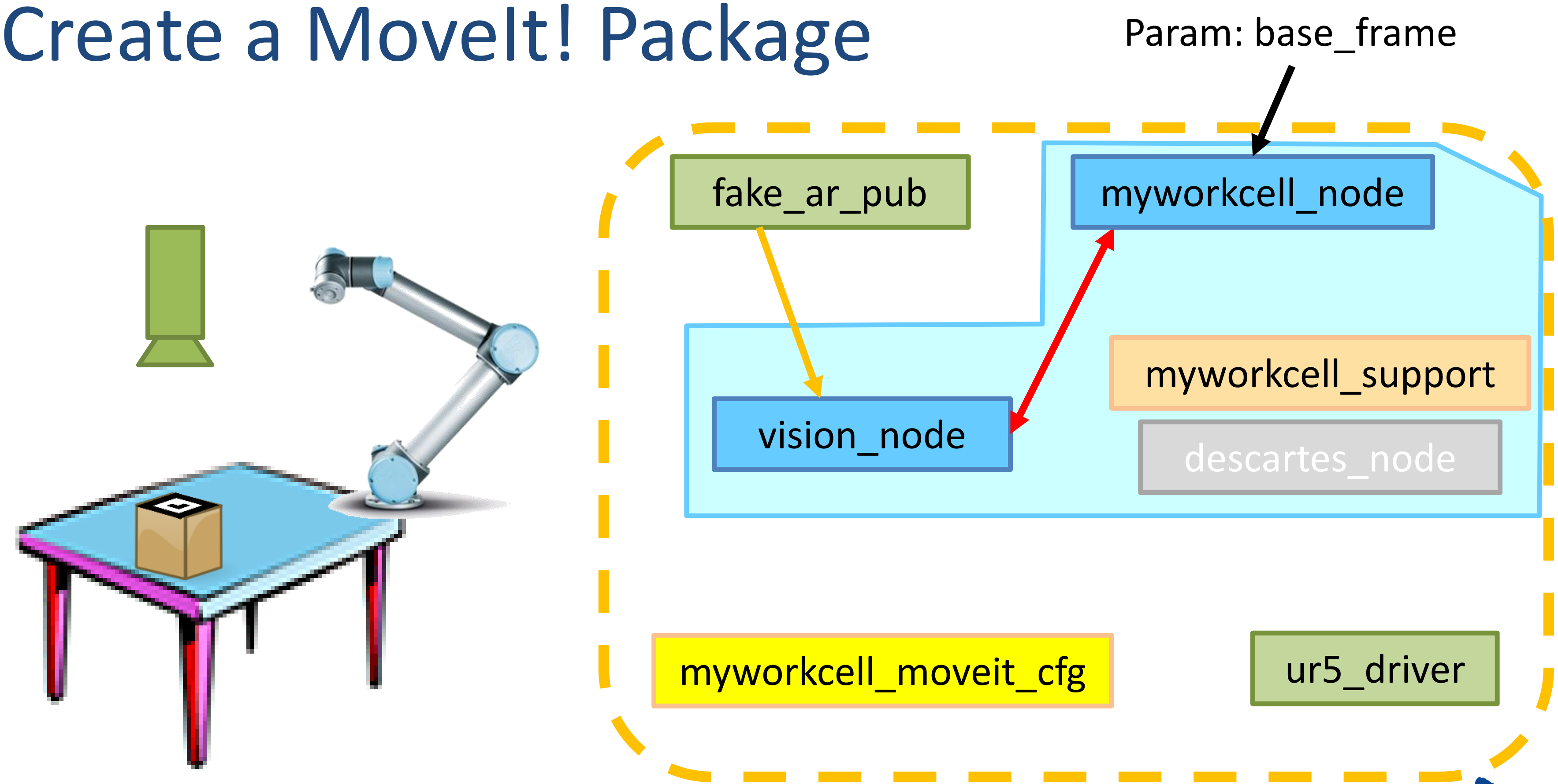
Update MoveIt! Package

- We'll generate launch files to run both:
  - **simulated** ROS-I robot
  - **real** robot-controller interface



Exercise 3.3:

Create a MoveIt! Package

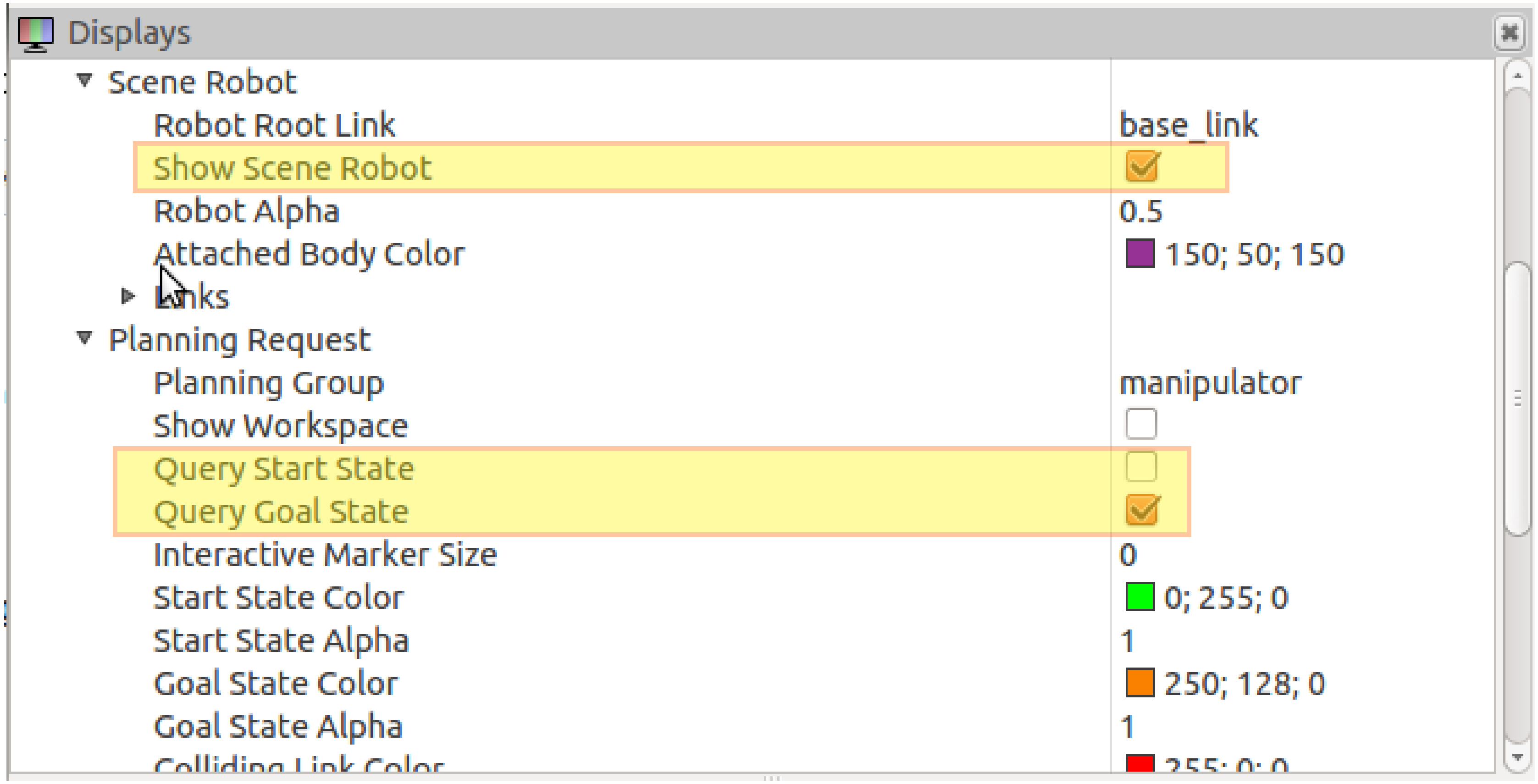


HowTo:
Motion Planning using MoveIt!

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

Motion Planning in RViz

Display Options

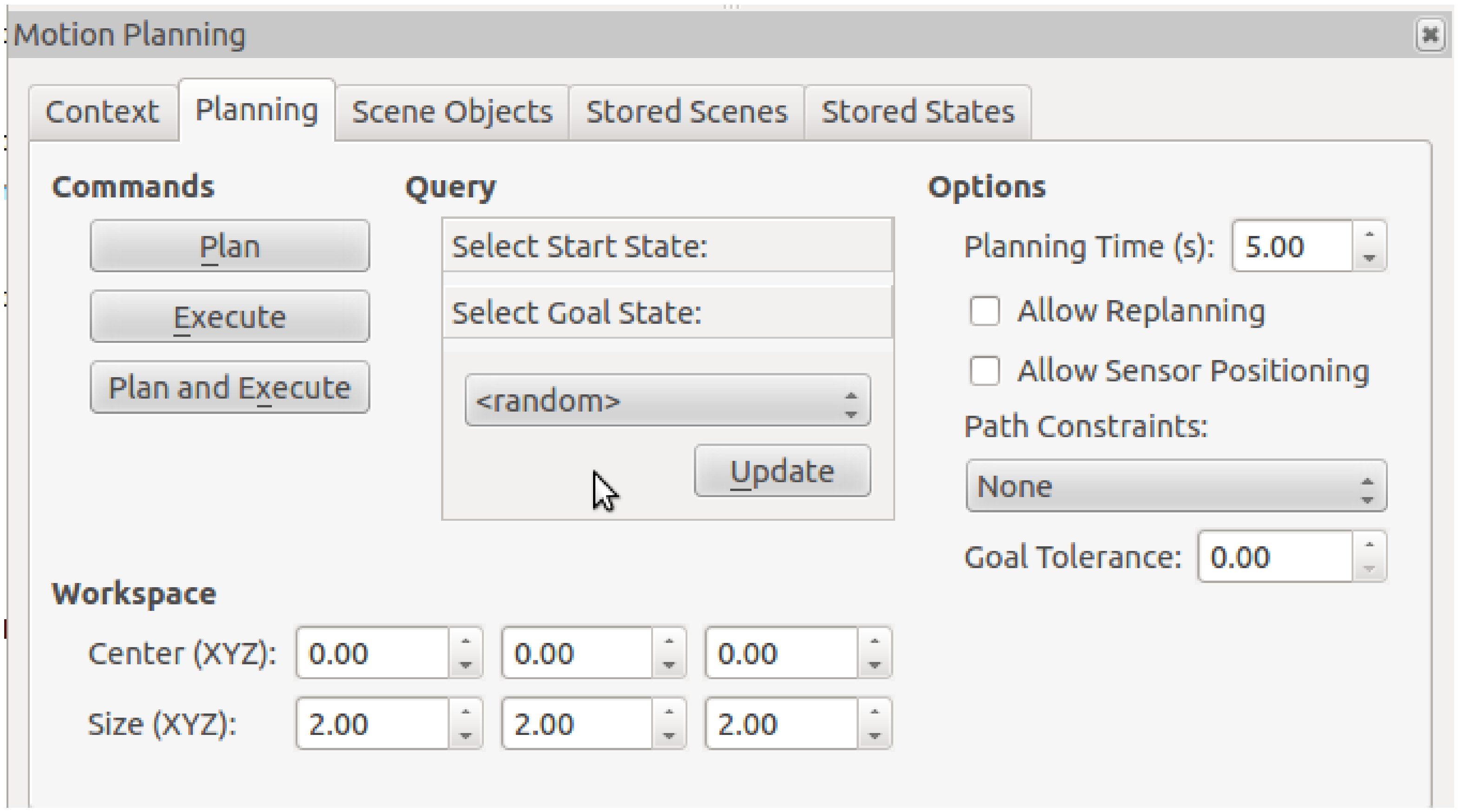


Displays

▼ Scene Robot	
Robot Root Link	base_link
Show Scene Robot	☑
Robot Alpha	0.5
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
Start State Alpha	1
Goal State Color	🟧 250; 128; 0
Goal State Alpha	1
Colliding Link Color	🟥 255; 0; 0

Motion Planning in RViz

Planning Options


Motion Planning

Context | Planning | Scene Objects | Stored Scenes | Stored States

Commands

Plan

Execute

Plan and Execute

Query

Select Start State:

Select Goal State:

<random>

Update

Options

Planning Time (s): 5.00

☐ Allow Replanning

☐ Allow Sensor Positioning

Path Constraints: None

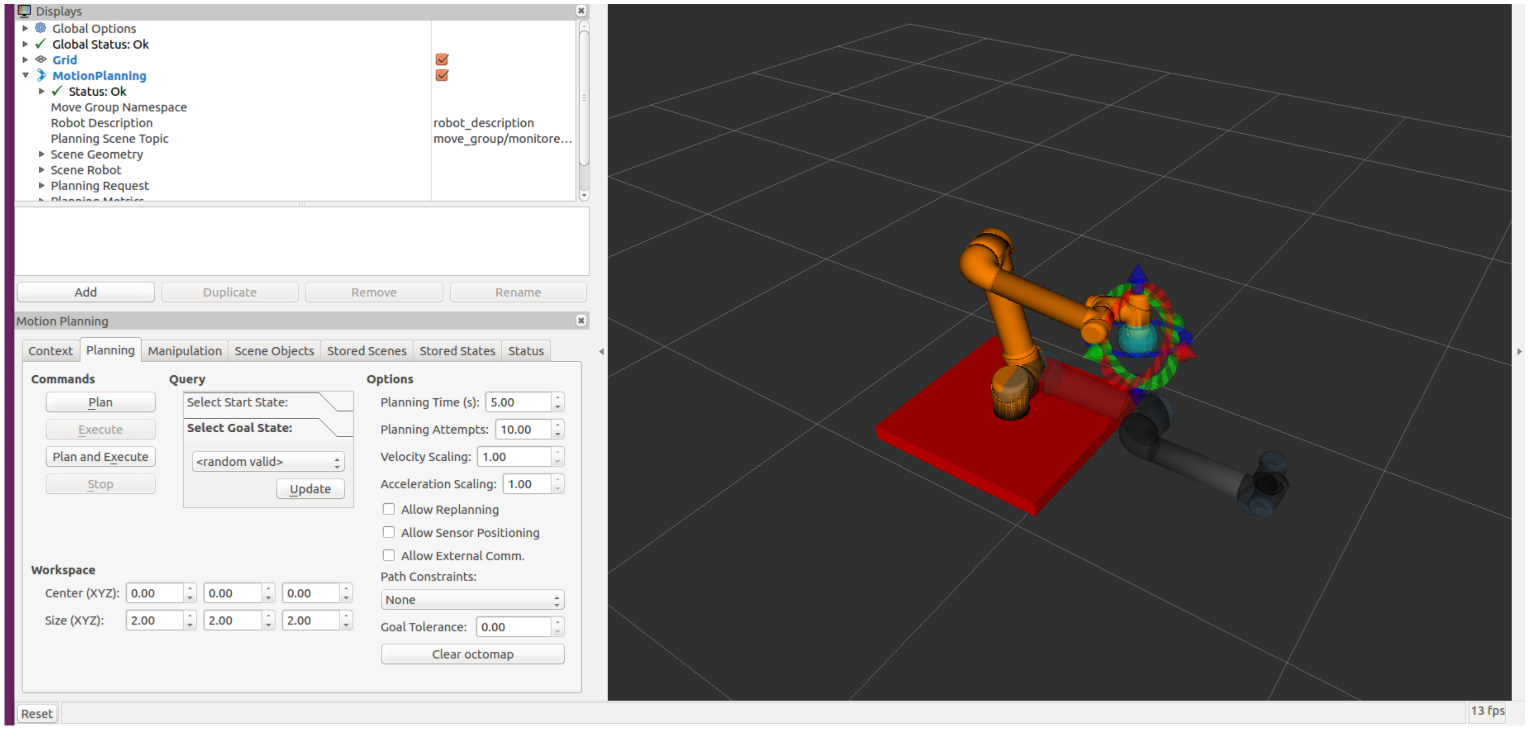
Goal Tolerance: 0.00

Workspace

Center (XYZ):	0.00	0.00	0.00
Size (XYZ):	2.00	2.00	2.00

Exercise 3.4:

Motion Planning using RViz



Displays
- Global Options
- ✓ Global Status: Ok
- Grid ✓
- MotionPlanning ✓
  - ✓ Status: Ok
  - Move Group Namespace
  - Robot Description: robot_description
  - Planning Scene Topic: move_group/monitore…
  - Scene Geometry
  - Scene Robot
  - Planning Request

Add | Duplicate | Remove | Rename

Motion Planning

Context | Planning | Manipulation | Scene Objects | Stored Scenes | Stored States | Status

Commands: Plan, Execute, Plan and Execute, Stop

Query
Select Start State:
Select Goal State:
<random valid>
Update

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.
Path Constraints: None
Goal Tolerance: 0.00
Clear octomap

Workspace
Center (XYZ): 0.00 0.00 0.00
Size (XYZ): 2.00 2.00 2.00

Reset 13 fps

Review

ROS

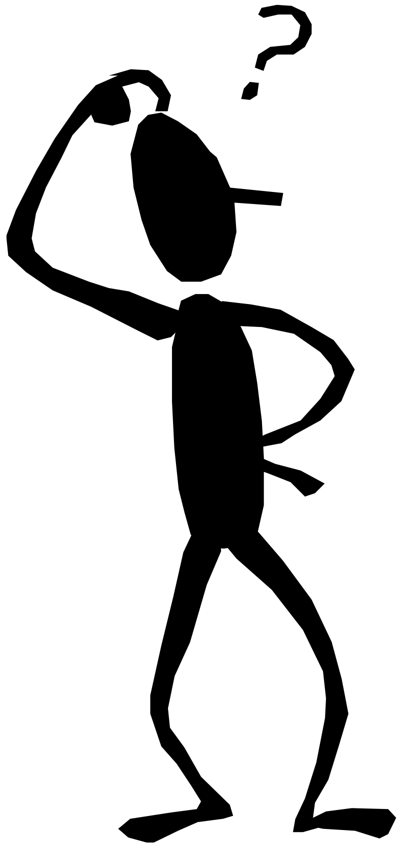
- URDF
- MoveIt
- Path Planners
- RViz Planning

ROS-Industrial

- Robot Drivers
- Path Planners

Questions?

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



Contact Info.

Jeremy Zoss

SwRI
9503 W. Commerce
San Antonio, TX 78227
USA

Phone: 210-522-3089
Email: jzoss@swri.org

Levi Armstrong

SwRI
9503 W. Commerce
San Antonio, TX 78227
USA

Phone: 210-522-3801
Email: levi.armstrong@swri.org