



ROS-Industrial Basic Developer's Training Class



Southwest Research Institute





Session 3:

Motion Control of Manipulators



Southwest Research Institute





URDF: Unified Robot Description Format

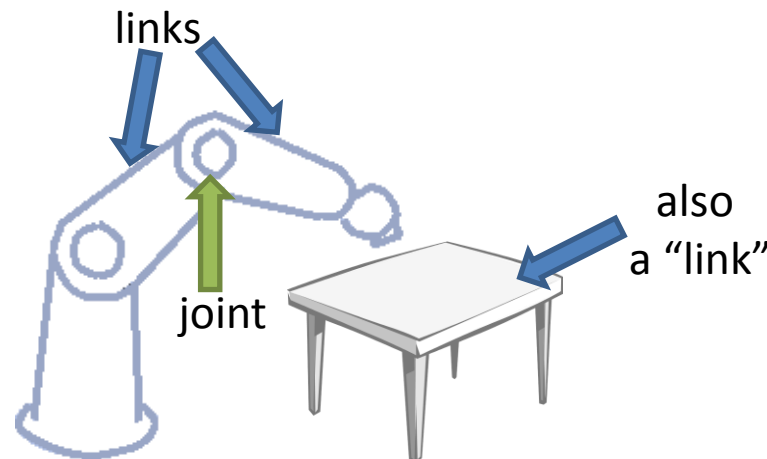




URDF: Overview



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



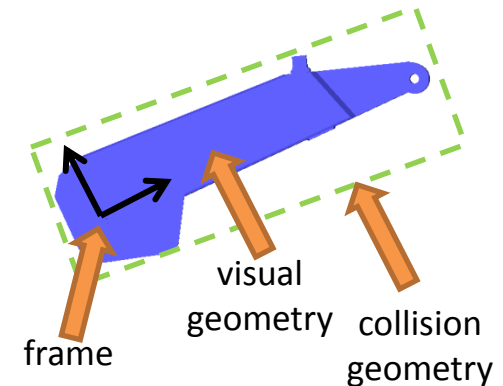


URDF: Link



- 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]

```
<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



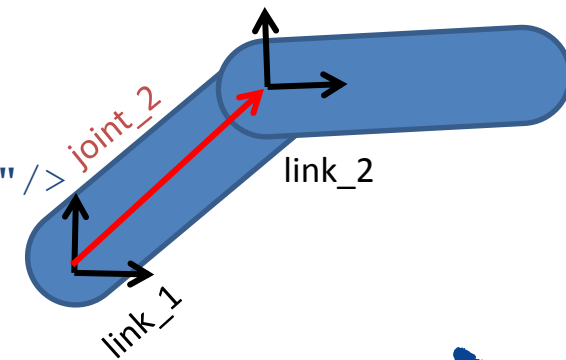


URDF: Joint



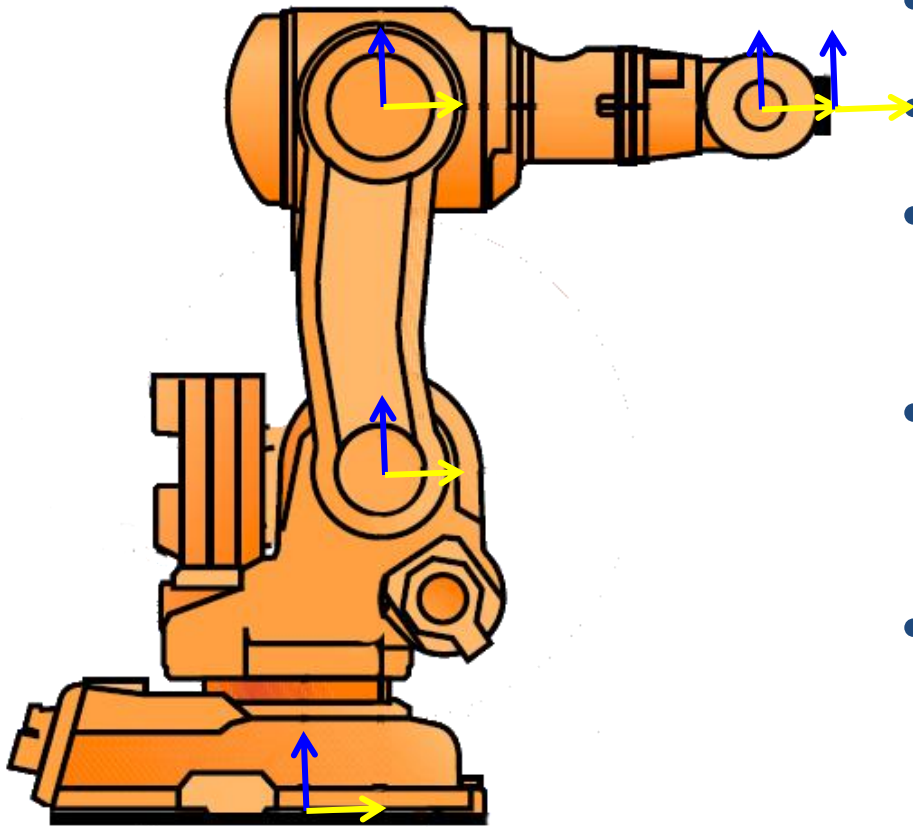
- A **Joint** connects two **Links**
 - Defines a **transform** between **parent** and **child** frames
 - Types: *fixed, free, linear, rotary*
 - Denotes axis of movement (*for linear / rotary*)
 - Contains joint limits on position and velocity

```
<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>
```





ROS-I Conventions



- Robot in Zero Position
- Place joints on axes
- Keep all frames same orientation
- X-Axis Front, Z-Axis Up
- *Unlike DH-Parameters, URDF allows free choice of frame orientation*



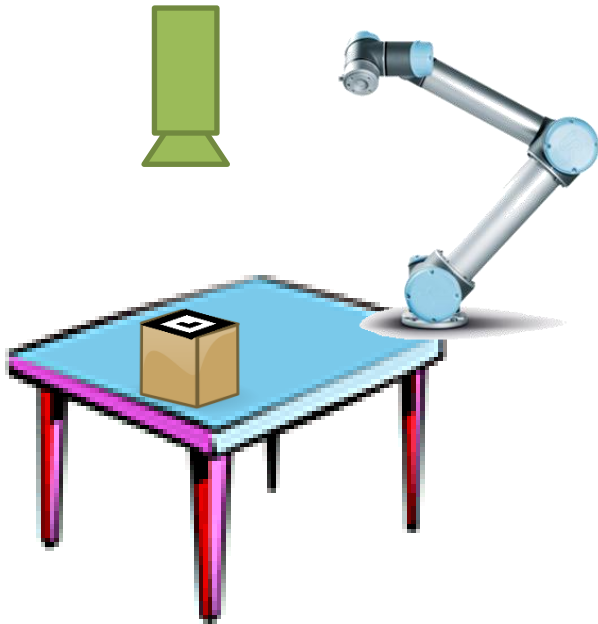


Exercise 3.0

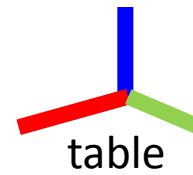
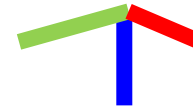


Exercise 3.0

Create a simple urdf



camera_frame





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>
```





XACRO -> URDF



- Most ROS tools expect URDFs, not XACRO
- Run the “xacro” command to convert XACRO files to URDF:

```
rosrun xacro xacro robot.xacro > robot.urdf
```

- Typically, xacro conversion is triggered by launch files, not executed manually.

```
<param name="robot_description"  
  command="$(find xacro)/xacro workcell.xacro" />
```





XACRO: Macros



Define
XACRO
Macro

robot.xacro

```
<xacro:macro name="robot" params="id">  
  <joint name="${id}_joint1">  
    ...  
</xacro>
```

Include
and
Call
Macro

workcell.xacro

```
<xacro:include filename="robot.xacro"/>  
  
<xacro:robot id:="left" />  
<xacro:robot id:="right" />
```





URDF Practical Examples



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



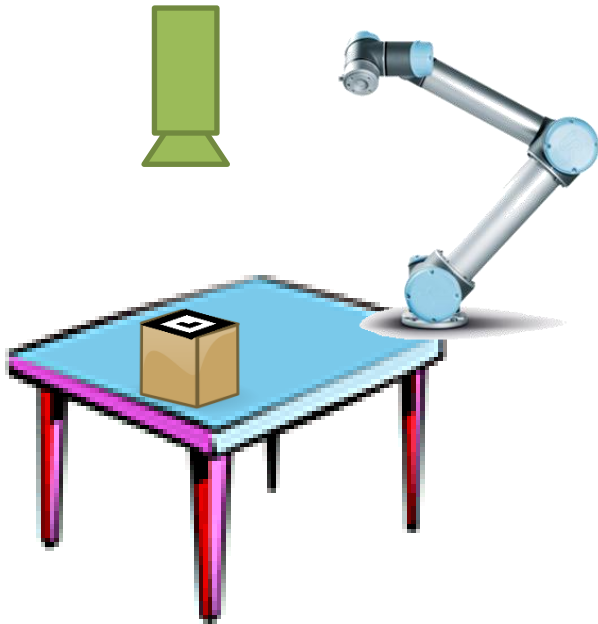


Exercise 3.1

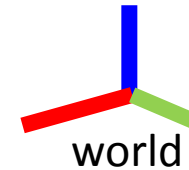
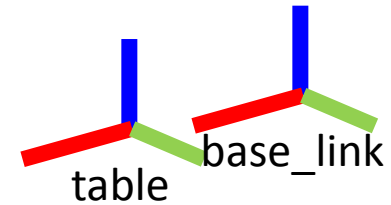
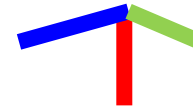


Exercise 3.1

Combine simple urdf with ur5 xacro



camera_frame





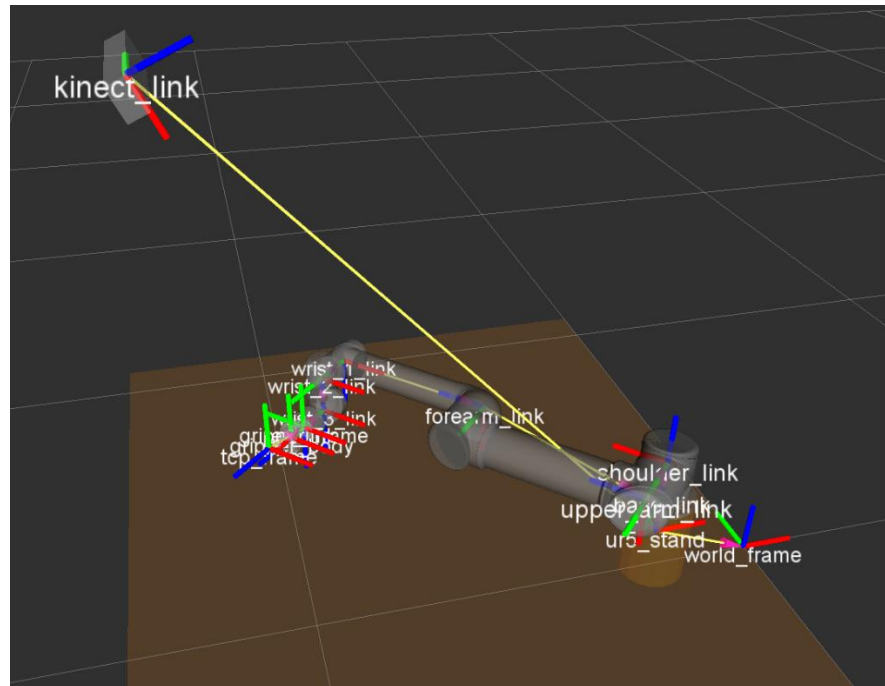
TF – Transforms in ROS





TF: Overview

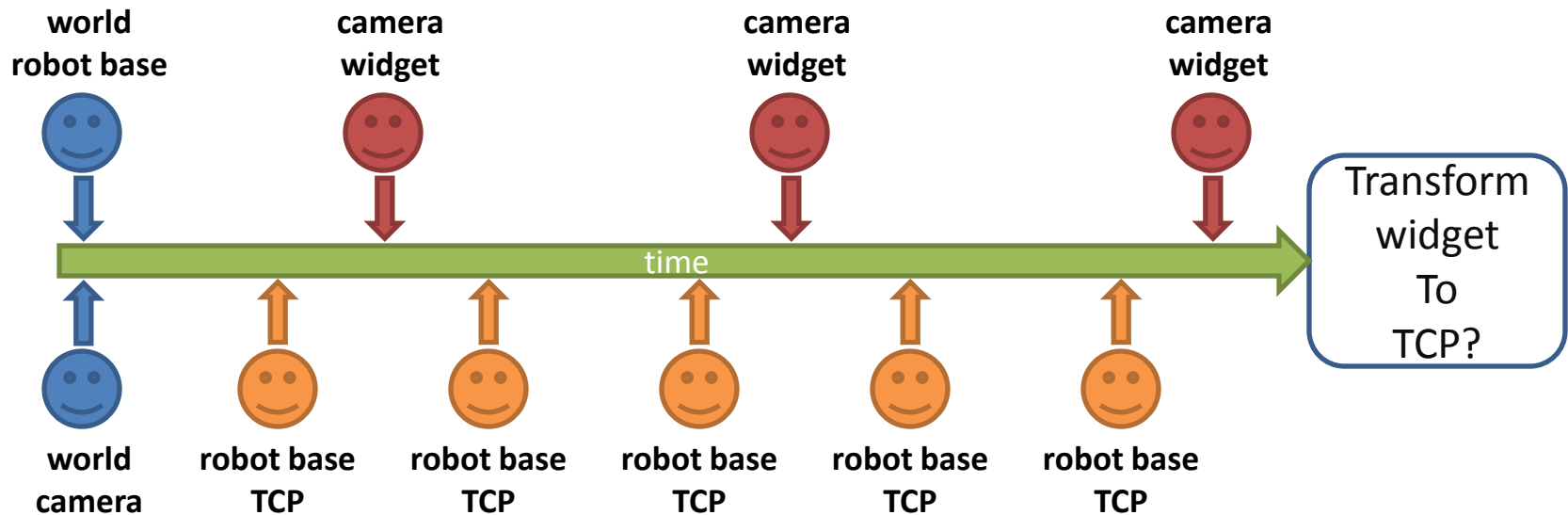
- TF is a **distributed framework** to track **coordinate frames**
- Each frame is related to at least one other frame





TF: Time Sync

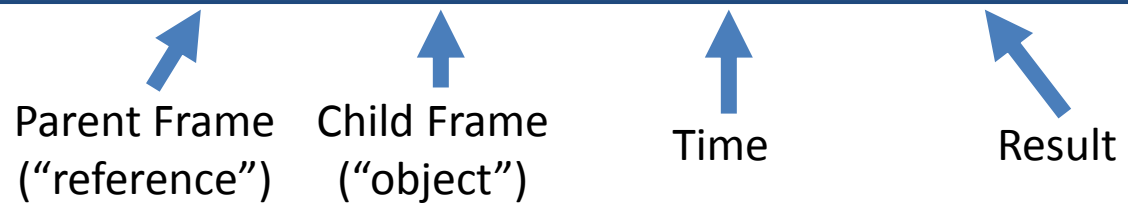
- TF tracks frame history
 - can be used to find transforms in the past!
 - essential for asynchronous / distributed system





- 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);
```



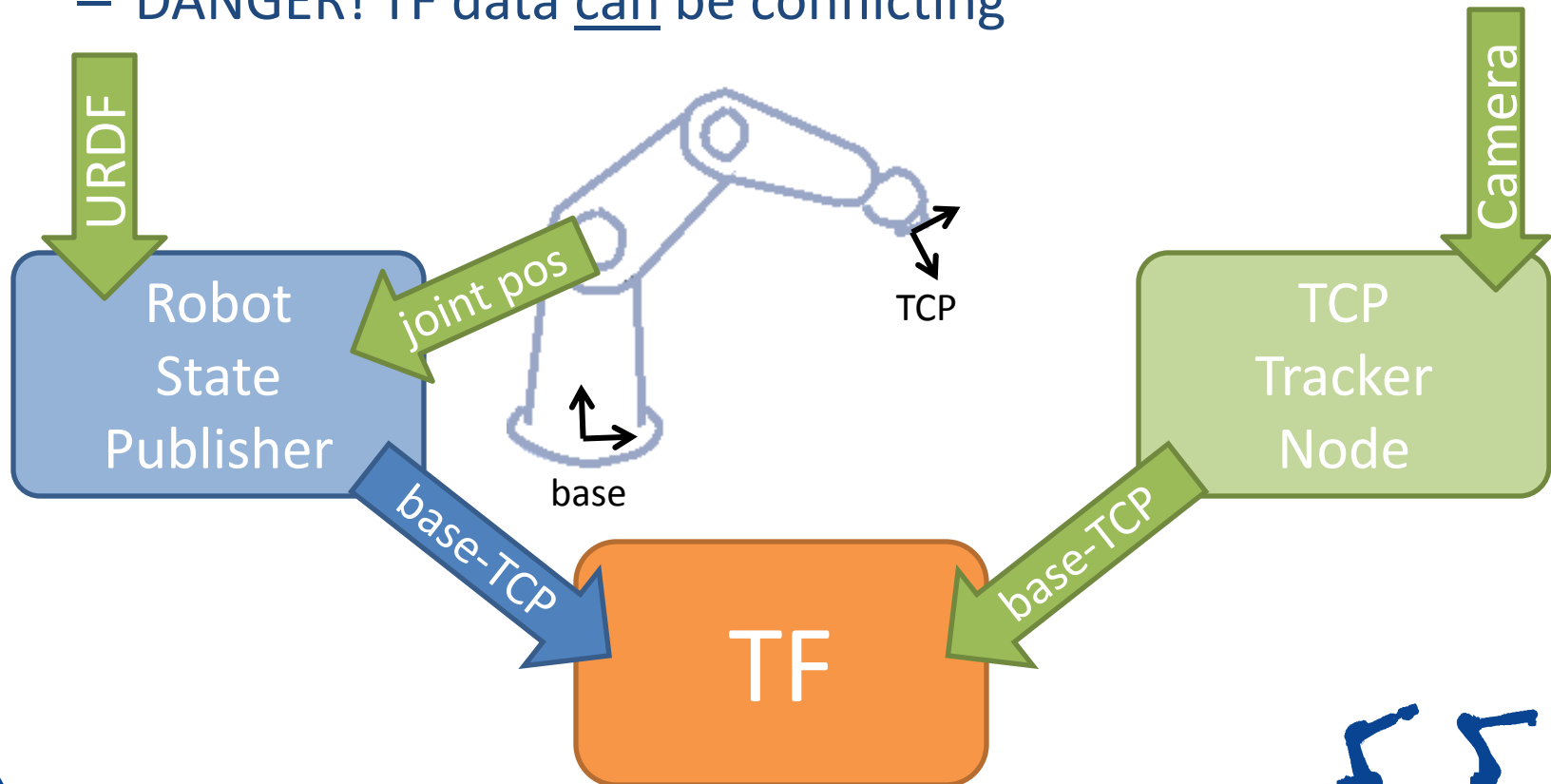
- Note confusing “target/source” naming convention
- `ros::Time()` or `ros::Time(0)` give **latest** available transform
- `ros::Time::now()` usually fails





TF: Sources

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



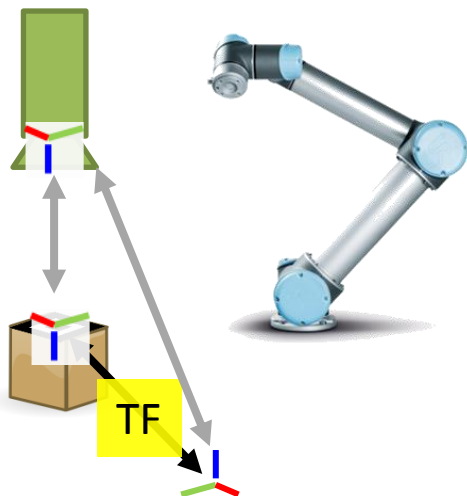


Exercise 3.2

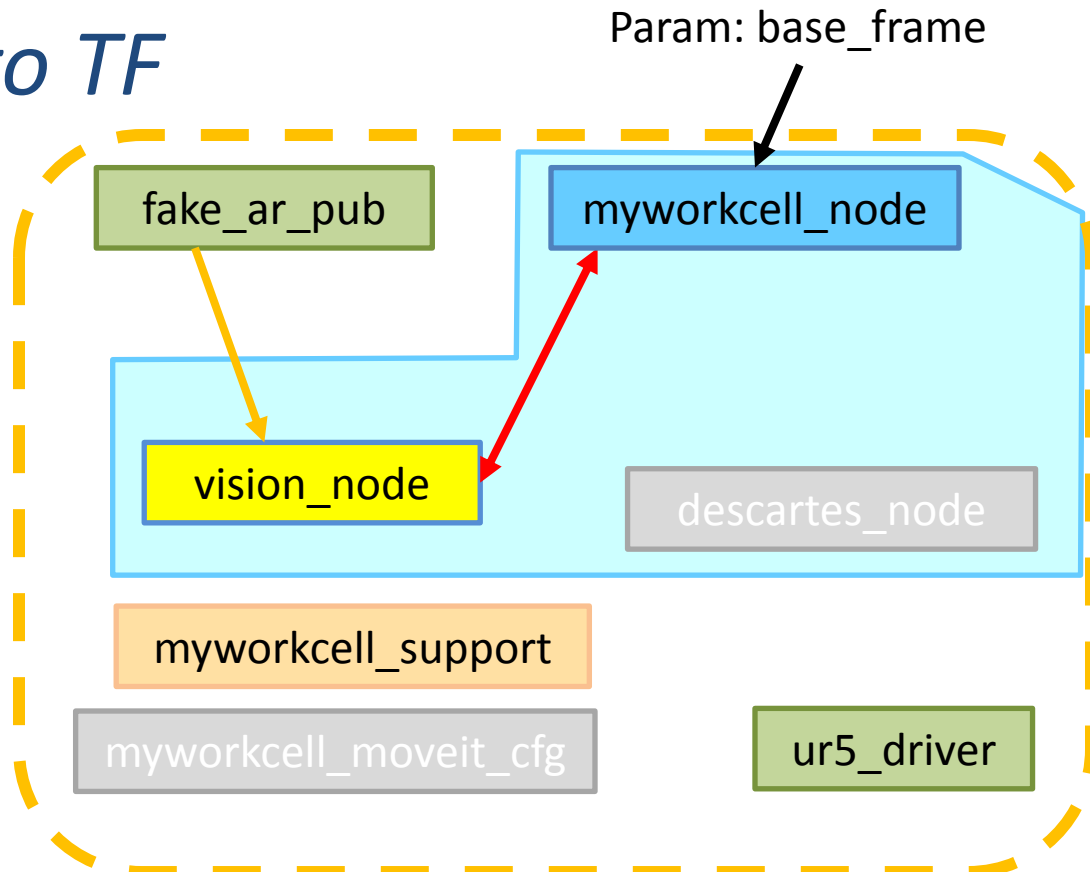


Exercise 3.2

Introduction to TF



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



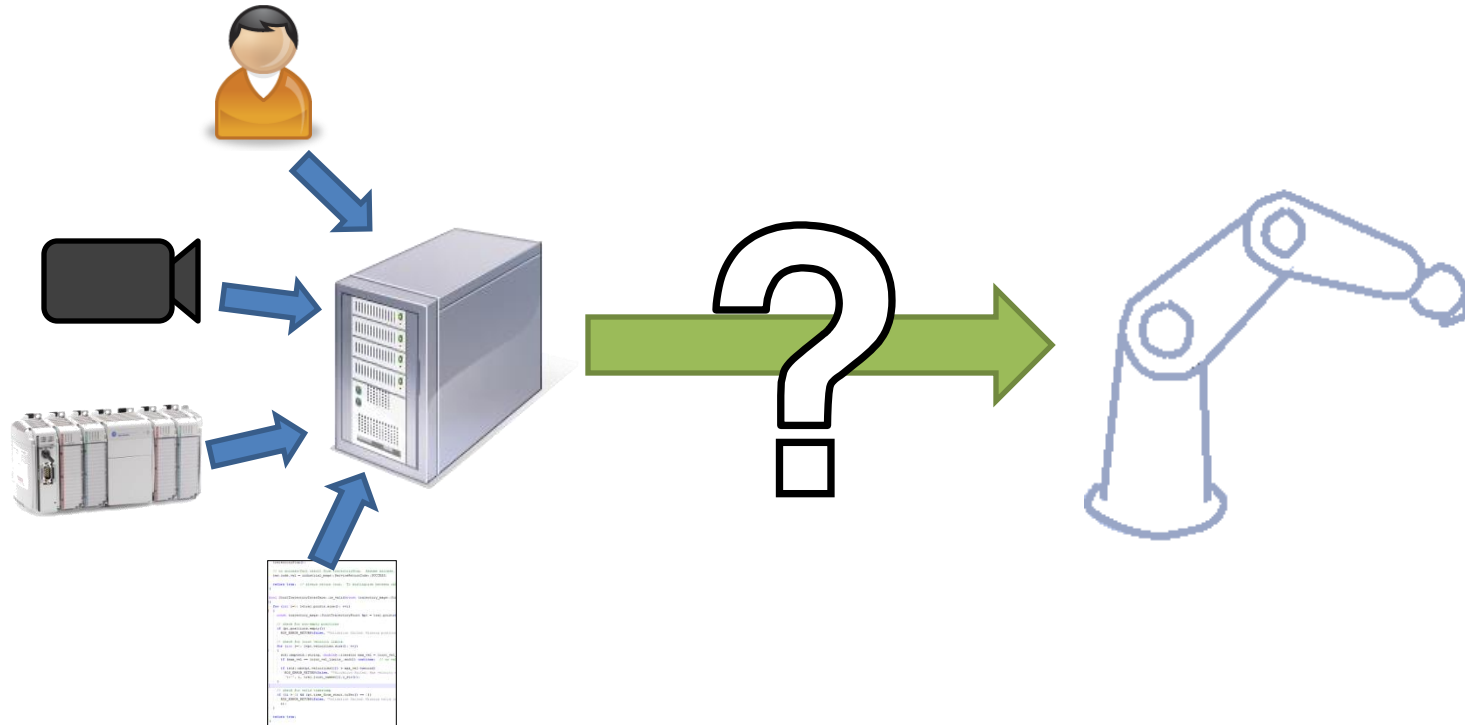


Motion Planning in ROS



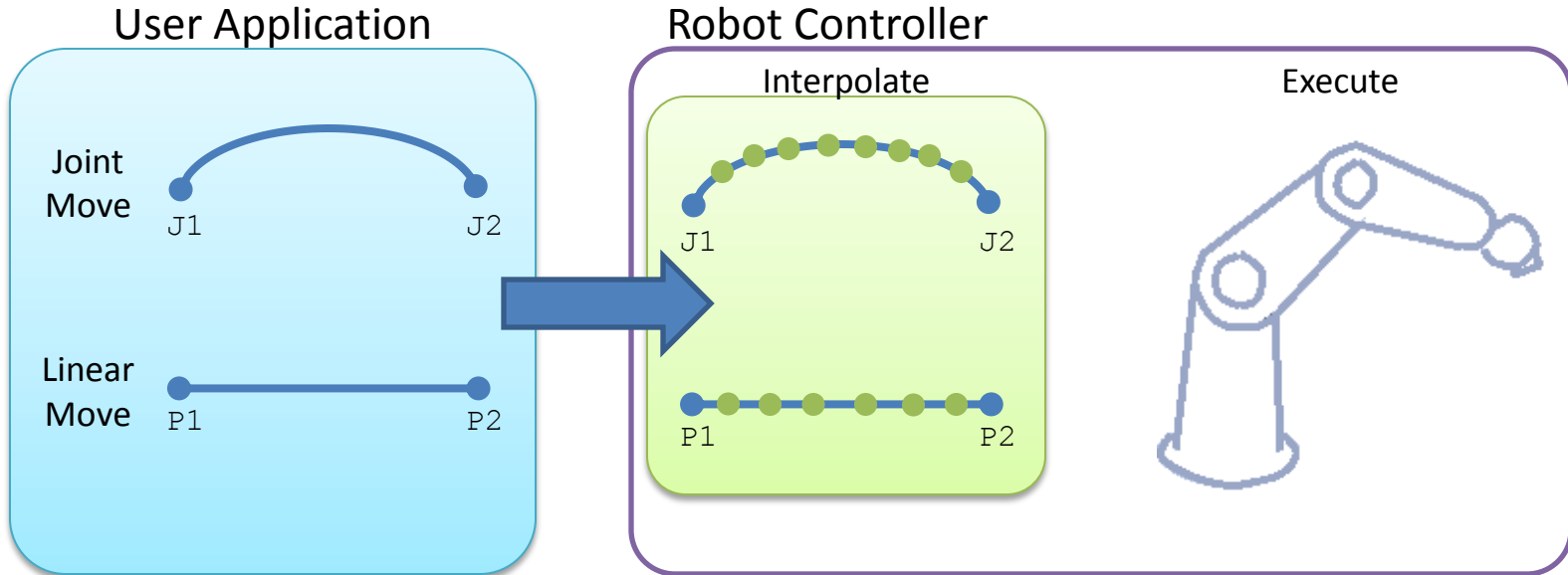


Motion Planning in ROS





Traditional Robot Programming

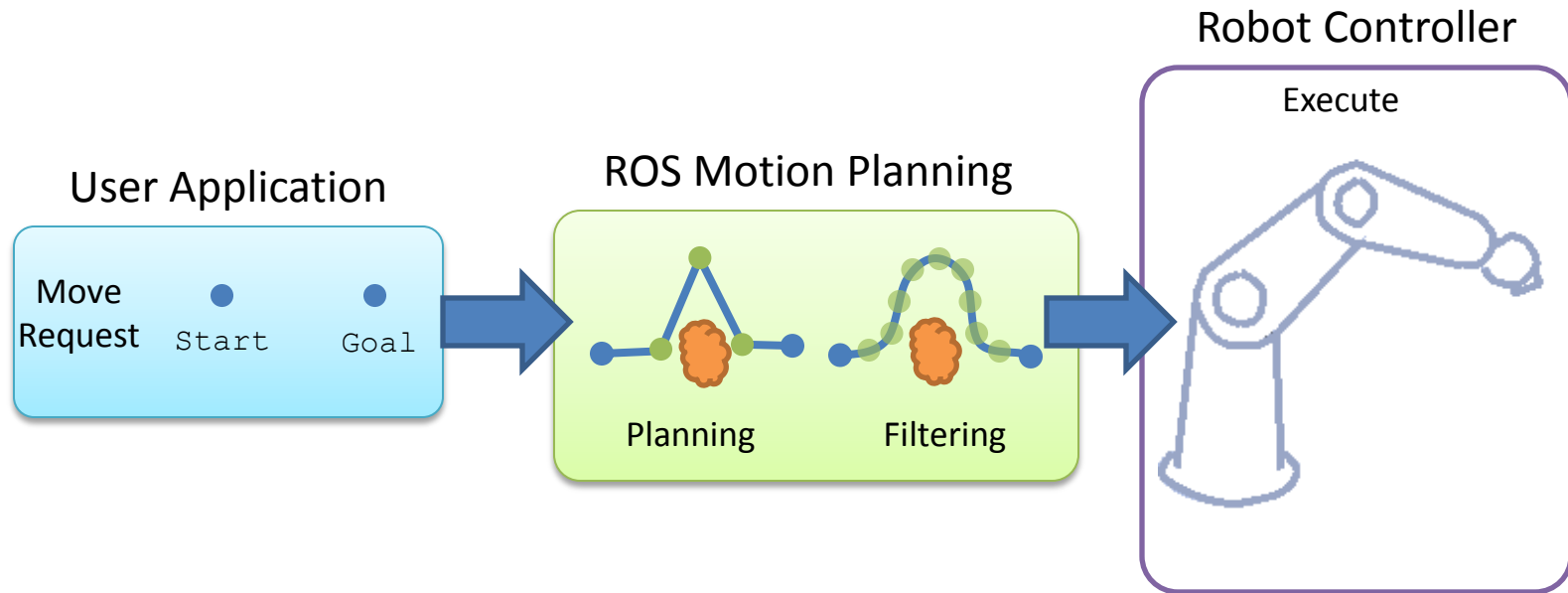


- **Motion Types:** *limited, but well-defined. One motion task.*
- **Environment Model:** *none*
- **Execution Monitor:** *application-specific*





Movelt Motion Planning

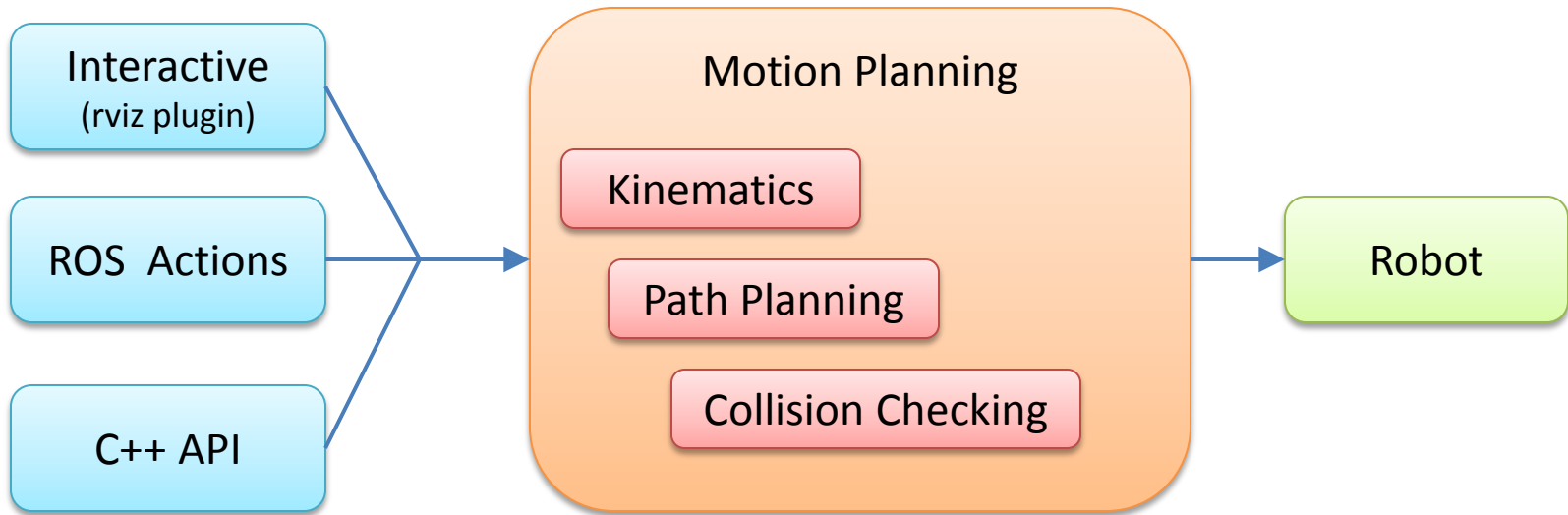


- Motion Types: *flexible, goal-driven, with constraints*
but minimal control over actual path
- Environment Model: *automatic, based on live sensor feedback*
- Execution Monitor: *detects changes during motion*



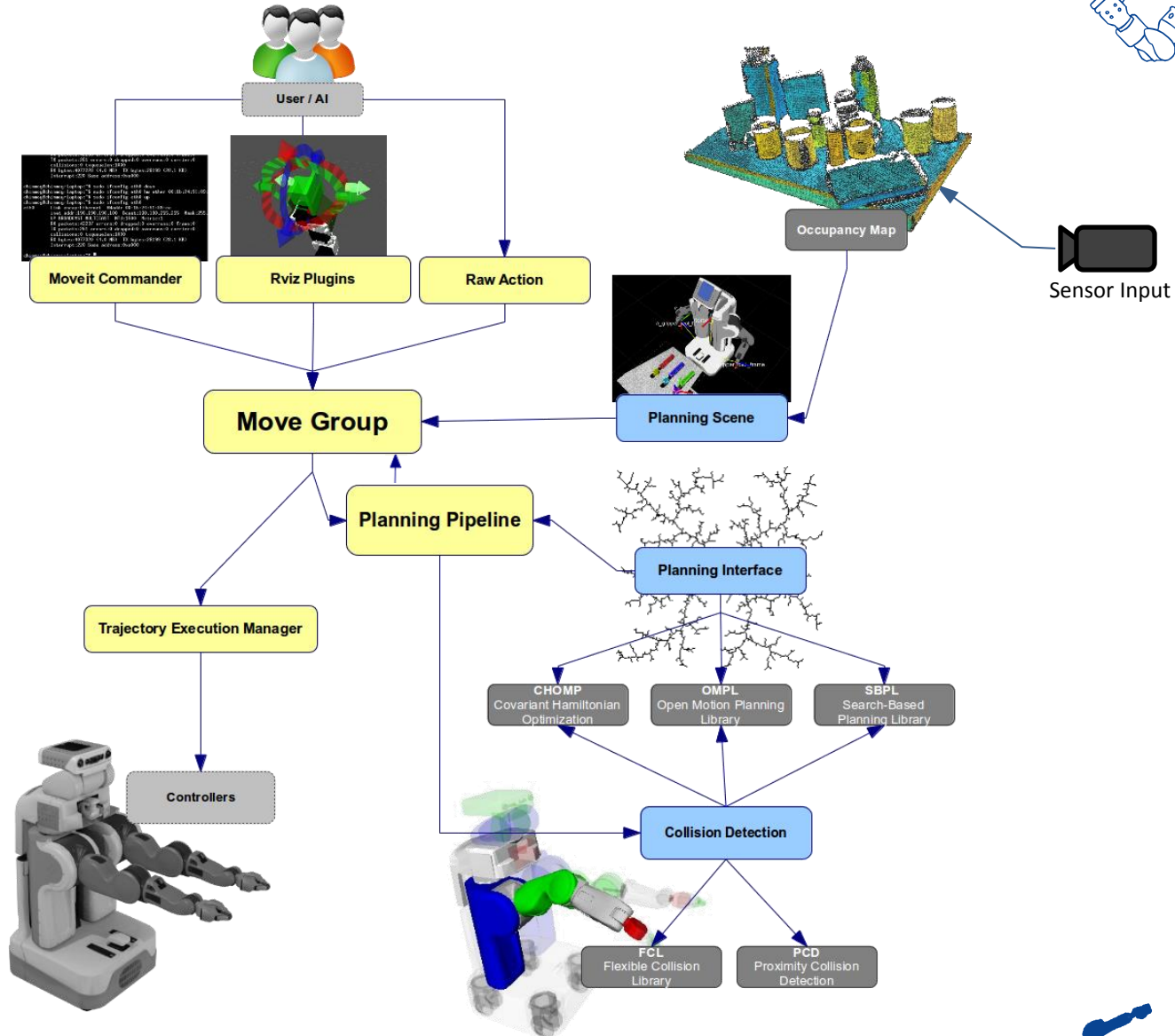


Motion Planning Components





Moveit Components

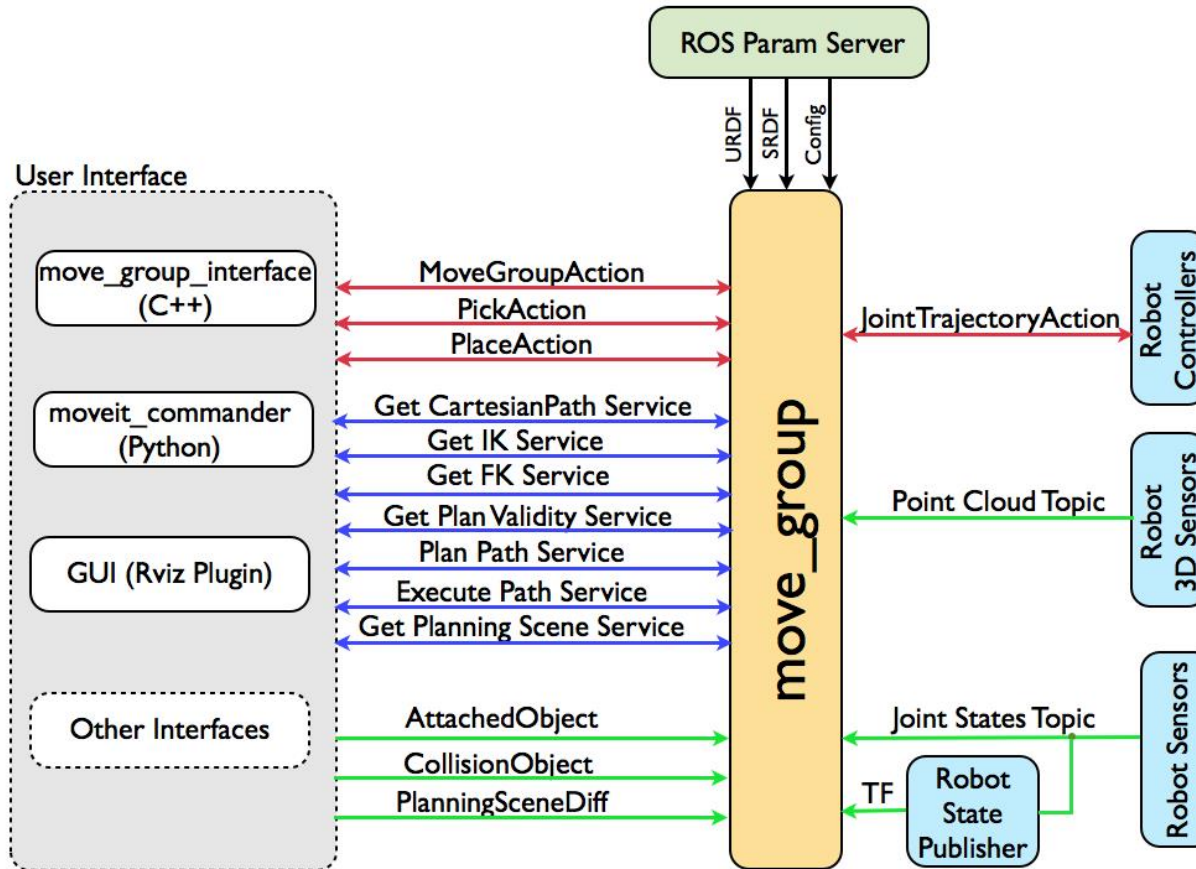


http://moveit.ros.org/wiki/High-level_Overview_Diagram
http://moveit.ros.org/wiki/Pipeline_Overview_Diagram





Movelt Nodes





- A Movelt! 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)





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

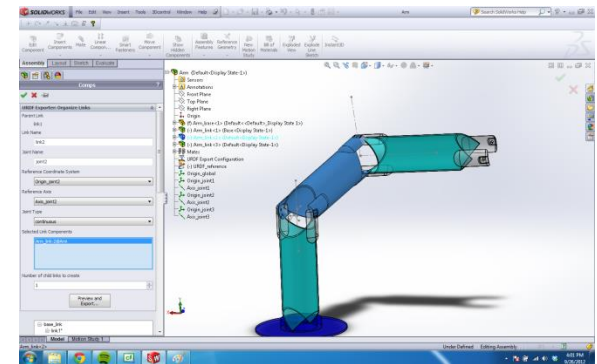




Create a URDF



- 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

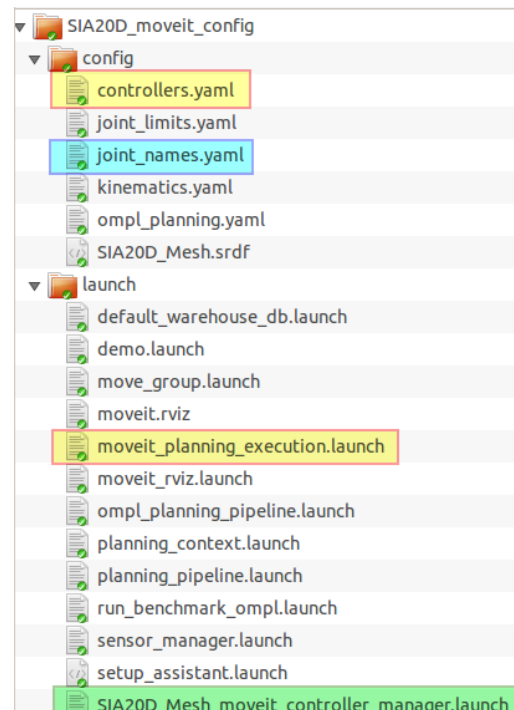




Update MoveIt! Package



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

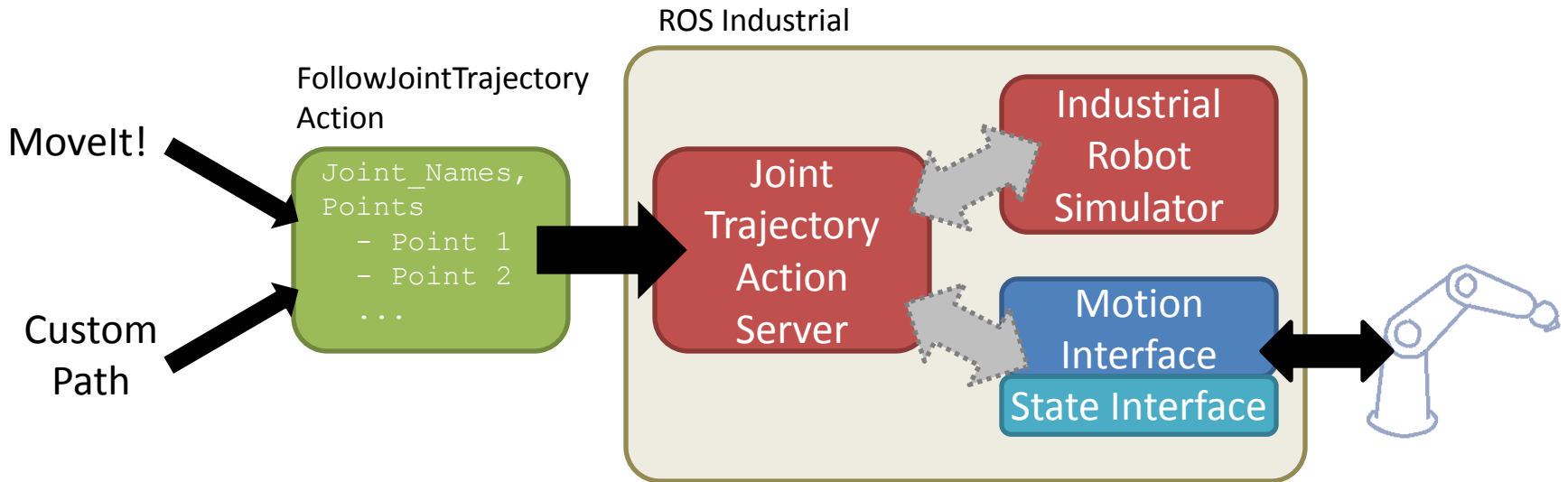




Update MoveIt! Package



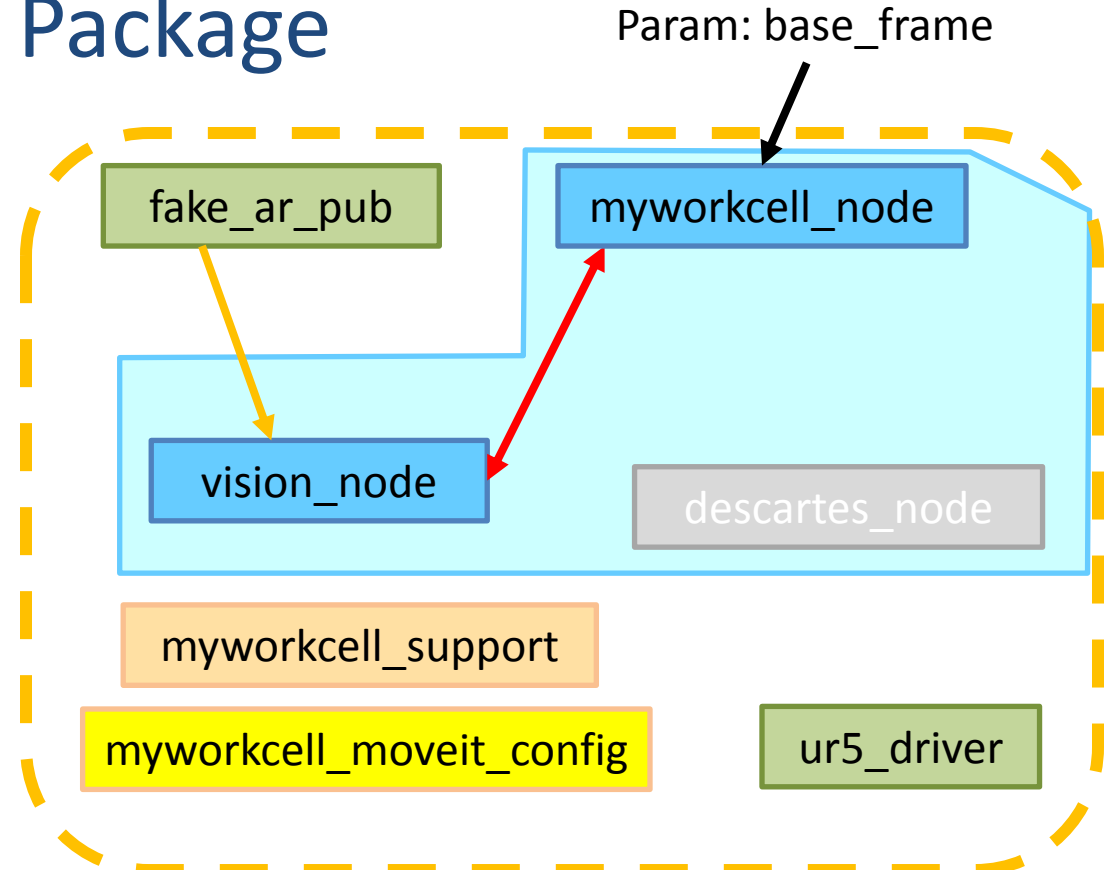
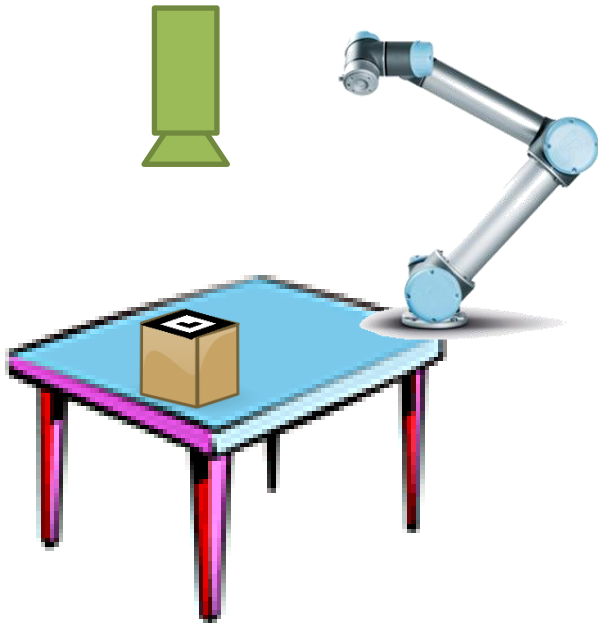
- We'll generate launch files to run both:
 - **simulated** ROS-I robot
 - **real** robot-controller interface





Exercise 3.3

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

The screenshot shows the 'Displays' panel in RViz. It is organized into two main sections: 'Scene Robot' and 'Planning Request'. The 'Scene Robot' section includes options for 'Robot Root Link' (set to 'base_link'), 'Show Scene Robot' (checked), 'Robot Alpha' (0.5), and 'Attached Body Color' (purple, 150; 50; 150). The 'Links' sub-section is expanded. The 'Planning Request' section includes 'Planning Group' (set to 'manipulator'), 'Show Workspace' (unchecked), 'Query Start State' (unchecked), 'Query Goal State' (checked), 'Interactive Marker Size' (0), 'Start State Color' (green, 0; 255; 0), 'Start State Alpha' (1), 'Goal State Color' (orange, 250; 128; 0), 'Goal State Alpha' (1), and 'Colliding Link Color' (red, 255; 0; 0).

Section	Option	Value
Scene Robot	Robot Root Link	base_link
	Show Scene Robot	<input checked="" type="checkbox"/>
	Robot Alpha	0.5
	Attached Body Color	150; 50; 150
	Links	Expanded
Planning Request	Planning Group	manipulator
	Show Workspace	<input type="checkbox"/>
	Query Start State	<input type="checkbox"/>
	Query Goal State	<input checked="" type="checkbox"/>
	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	





Planning Options

The screenshot shows the 'Motion Planning' window with the 'Planning' tab selected. The interface is divided into several sections:

- Commands:** Three buttons labeled 'Plan', 'Execute', and 'Plan and Execute'.
- Query:** Two dropdown menus for 'Select Start State' and 'Select Goal State'. The 'Select Goal State' dropdown is currently set to '<random>'. Below these is an 'Update' button.
- Options:** A 'Planning Time (s)' dropdown set to '5.00'. Two checkboxes: 'Allow Replanning' (unchecked) and 'Allow Sensor Positioning' (unchecked). A 'Path Constraints' dropdown set to 'None'. A 'Goal Tolerance' dropdown set to '0.00'.
- Workspace:** Two rows of three numerical input fields. The first row is labeled 'Center (XYZ):' with values '0.00', '0.00', and '0.00'. The second row is labeled 'Size (XYZ):' with values '2.00', '2.00', and '2.00'.

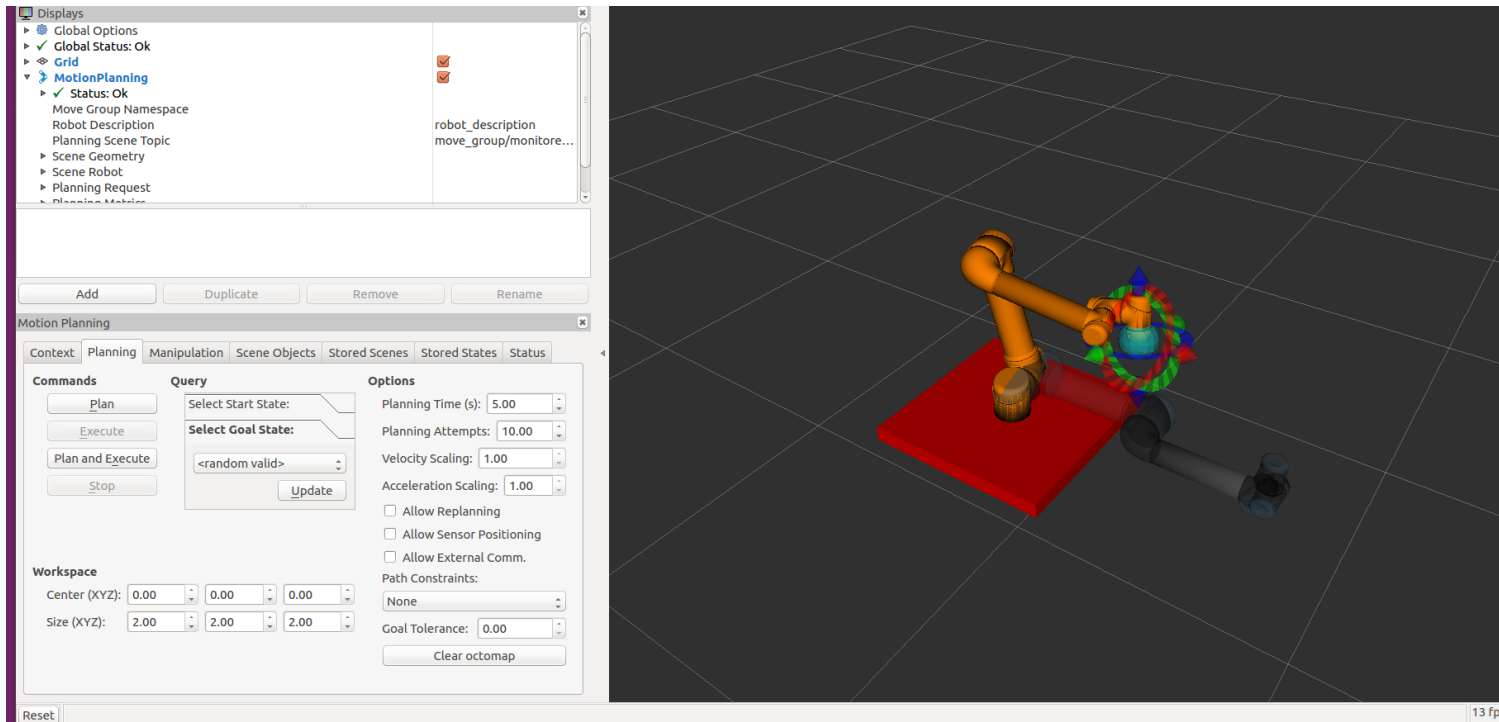




Exercise 3.4



Exercise 3.4: Motion Planning using RVIZ





ROS

- URDF
- MoveIt
- Path Planners
- RViz Planning

ROS-Industrial

- Robot Drivers
- Path Planners





Questions?



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

