TECHIN 516: Robotic Sensing And Mobility

Lab 5: Introduction to Robotic Arm Kinematics

DATE: Feb 16th. 2023

Lecturer: Queenie Qiu, John Raiti. Student: ***Shucheng Guo***

1 Learning Objectives

1. Learn the difference between arm motions in task-space and configuration-space.
2. Familiarize yourself with MoveIt! as a ROS component.
3. Implement high-level nodes through MoveIt! combined with RViz and Gazebo to move a Kinova Gen3lite arm and a Fetch robotics arm using forward and inverse kinematics.

2 Introducing Kinova Gen3lite

2.1 ROS Kortex Package Installation

ROS Kortex is the official ROS package to interact with Kortex and its related products. These are the instructions to run in a terminal to clone the ros_kortex repository and install the necessary ROS dependencies.
Note:The default branch for git clone is developed for ROS Noetic. If you are using other ROS distributions, please git clone to corresponding branch.

```
$ sudo apt install python3 python3-pip
$ sudo python3 -m pip install conan
$ conan config set general.revisions_enabled=1
$ conan profile new default --detect > /dev/null
$ conan profile update settings.compiler.libcxx=libstdc++11 default
$ cd catkin_ws/src
$ git clone https://github.com/Kinovarobotics/ros_kortex.git
$ cd ../
$ rosdep install --from-paths src --ignore-src -y
```

Then, to build and source the workspace:

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

2.2 Spawning a Kinova Gen3lite robot in Gazebo

We will use the ros_kortex repository to work with the Kinova arm. In this metapackage, we will use the kortex_gazebo portion for simulation. Execute the following launch file that will bring up a gen3lite in Gazebo and open an RViZ window

```
$ roslaunch kortex_gazebo spawn_kortex_robot.launch arm:=gen3_lite z0:=0.0
```

The Gazebo window looks like:

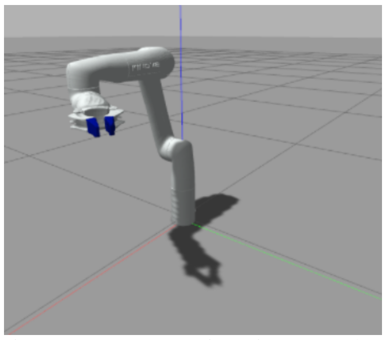
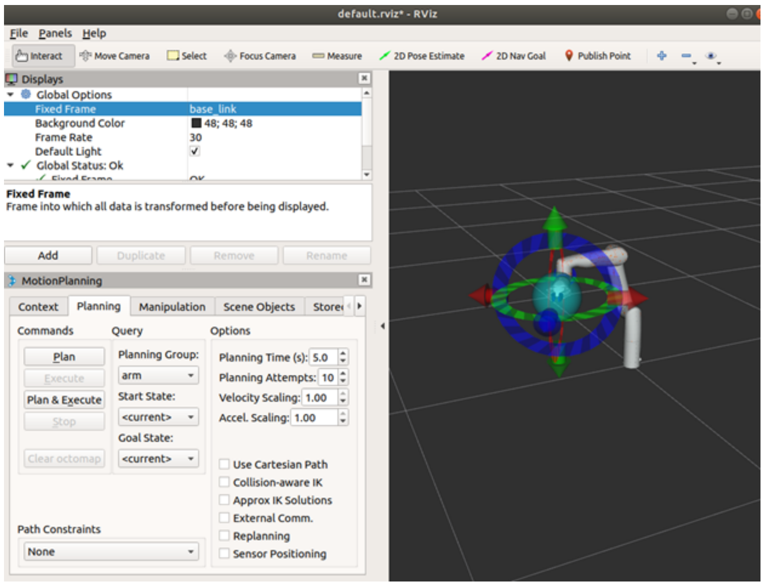


Figure 1: Spawning in Gazebo.

Note: Once the RViZ and Gazebo windows have been opened, and the terminal shows two green messages (“You can start planning now!” and “The Kortex driver has been initialized correctly!”) you can add a Robot model and add a MotionPlanning component in Rviz.



Global Options
Fixed Frame: base_link
Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
Global Status: Ok

Fixed Frame
Frame into which all data is transformed before being displayed.

Planning Group: arm
Start State: <current>
Goal State: <current>
Path Constraints: None

Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00

☐ Use Cartesian Path
☐ Collision-aware IK
☐ Approx IK Solutions
☐ External Comm.
☐ Replanning
☐ Sensor Positioning

Figure 2: Kinova arm in Rviz.

2.3 Using the Interactive Markers to change the robot’s pose based on the task-space

You can use the different arrows and rings on the interactive marker, to change positions and orientations of the robot’s end-effector. As you use them, you will see an orange version of the robot arm with the proposed new position. Once you have reached a desired position, use the buttons “Plan” and “Execute”(in the MotionPlanning Panel) to make the robot in Gazebo move to the new proposed position. Some useful tools for you to track the robot’s motions are:

1. Check the values of the joints:

```
$ rostopic echo -n 1 /my_gen3_lite/joint_states
```

2. Outputs the pose of the robot's end-effector with respect to the world (in this case Gazebo's origin):

```
$ rosrun tf tf_echo /world /end_effector_link
```

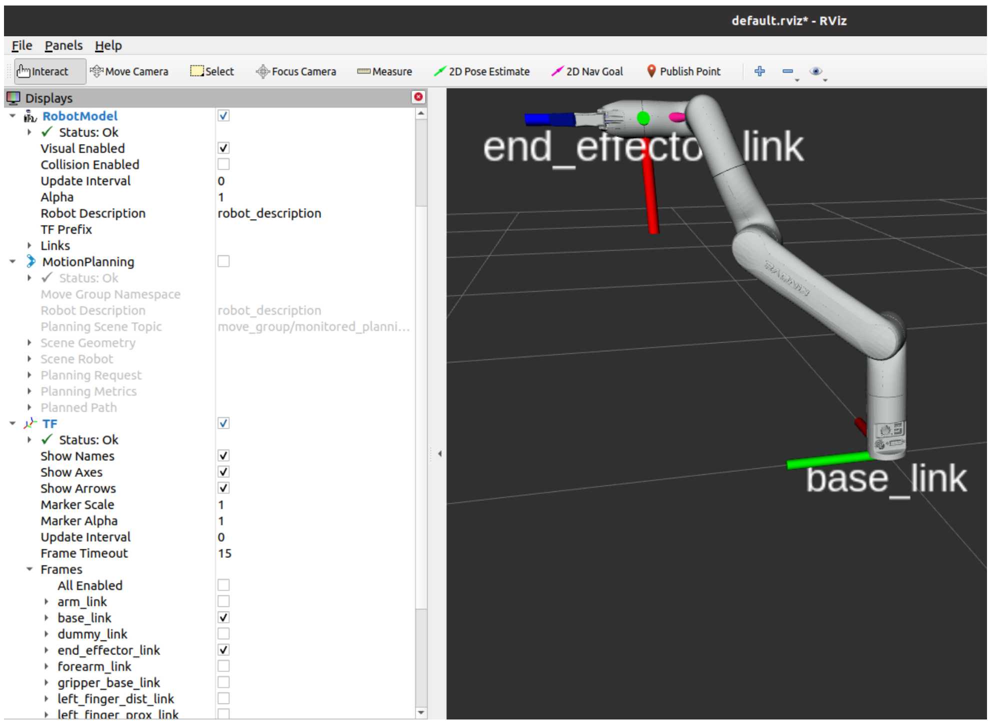
3. You may also consider using the base link as the reference:

```
$ rosrun tf tf_echo /base_link <END-EFFECTOR LINK>
```

If you want to change the state of the gripper, you need to change the Planning Group (in the MotionPlanning Panel) from arm to gripper, and you can select a new position.

Deliverables:

1. Add a TF component in RViz and enable only the base_link and the end_effector_link. Disable the Motion Planning component to temporarily remove the interactive markers. Add a screenshot of the RViZ view.


default.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
end_effecto link
base_link

RobotModel	✓
Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
MotionPlanning	☐
Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	move_group/monitored_planni…
Scene Geometry	
Scene Robot	
Planning Request	
Planning Metrics	
Planned Path	
TF	✓
Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	☐
arm_link	☐
base_link	✓
dummy_link	☐
end_effector_link	✓
forearm_link	☐
gripper_base_link	☐
left_finger_dist_link	☐
left_finger_prox_link	☐

Figure 3: RViZ with Kinova.

2. Use rqt_tf_tree to determine the kinematic chain between the two reference frames (base_link and end_effector_link).

Answer: According to the flowchart shown in `rqt_tf_tree`, the kinematic chain between `base_link` and `end_effector_link` can be represented as:

world $\rightarrow$ *base_link* $\rightarrow$ *shoulder_link* $\rightarrow$ *arm_link* $\rightarrow$ *forearm_link* $\rightarrow$ *lower_wrist_link* $\rightarrow$ *upper_wrist_link* $\rightarrow$ *end_effector_link* $\rightarrow$ *dummy_link*

3. Turn on the Motion Planning component again. The "Goal State" in the MotionPlanning panel allows you to set goals to some predefined positions. Inspect it and report what are the reported robot's joints values and pose of the end effector (wrt the world) when you send the gen3lite to the following preset positions:

   (a) Home

Table 1: Pose information of end effector in home state

(a) Position

Linear	*x*	*y*	*z*
Position	0.37	0.08	0.45

(b) Orientation

Angular	*x*	*y*	*z*	*w*
Orientation	-0.35	0.62	0.36	0.61

Table 2: Joint angles of end effector in home state

Joint	*1*	*2*	*3*	*4*	*5*	*6*
Angle	0	-16	75	0	-60	0

   (b) Retract

Table 3: Pose information of end effector in retract state

(a) Position

Linear	*x*	*y*	*z*
Position	0.13	-0.07	0.27

(b) Orientation

Angular	*x*	*y*	*z*	*w*
Orientation	0.99	-0.06	0.14	-0.01

Table 4: Joint angles of end effector in retract state

Joint	*1*	*2*	*3*	*4*	*5*	*6*
Angle	-3	21	145	-88	-40	-87

(c) Select a robot pose that you would choose if you had to pick something from the ground. Add a screenshot of the resulting pose of the arm in **Gazebo**.

Table 5: Pose information of end effector in pickup state

(a) Position

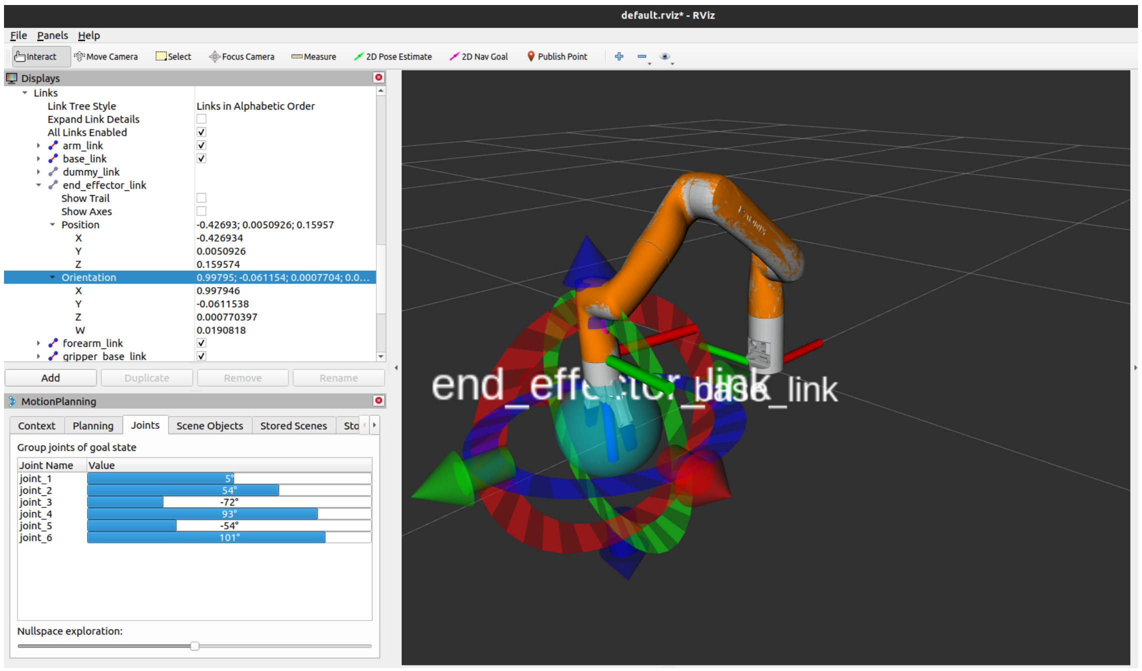
Linear	*x*	*y*	*z*
Position	-0.43	0.01	0.16

(b) Orientation

Angular	*x*	*y*	*z*	*w*
Orientation	1.00	-0.06	0.00	0.02

Table 6: Joint angles of end effector in pickup state

Joint	*1*	*2*	*3*	*4*	*5*	*6*
Angle	5	54	-72	93	-54	101



Links	
Link Tree Style	Links in Alphabetic Order
Expand Link Details	☐
All Links Enabled	✓
arm_link	✓
base_link	✓
dummy_link	
end_effector_link	
Show Trail	☐
Show Axes	☐
Position	-0.42693; 0.0050926; 0.15957
X	-0.426934
Y	0.0050926
Z	0.159574
Orientation	0.99795; -0.061154; 0.0007704; 0.0…
X	0.997946
Y	-0.0611538
Z	0.000770397
W	0.0190818
forearm_link	✓
gripper_base_link	✓

Group joints of goal state

Joint Name	Value
joint_1	5°
joint_2	54°
joint_3	-72°
joint_4	93°
joint_5	-54°
joint_6	101°

Nullspace exploration:

Figure 4: Picking up state in RViz from ground with Kinova arm.

2.4 Using the Joints tab in MotionPlanning to change the robot's pose on the configuration-space

In the MotionPlanning panel, find the "Joints" tab. You will be able to change the values of the different joint angles. Explore the configuration space, determine which joints move in which direction when a positive or a negative angle is given to each joint.

Deliverables:

1. Configure the joint angles and move to your desired position. Add a screenshot of the rviz window(including joint values and the robot visualization), use the same tools described in the previous subsection to explore the values for the pose and the joint angles.

Table 7: Pose information of end effector in custom state

(a) Position

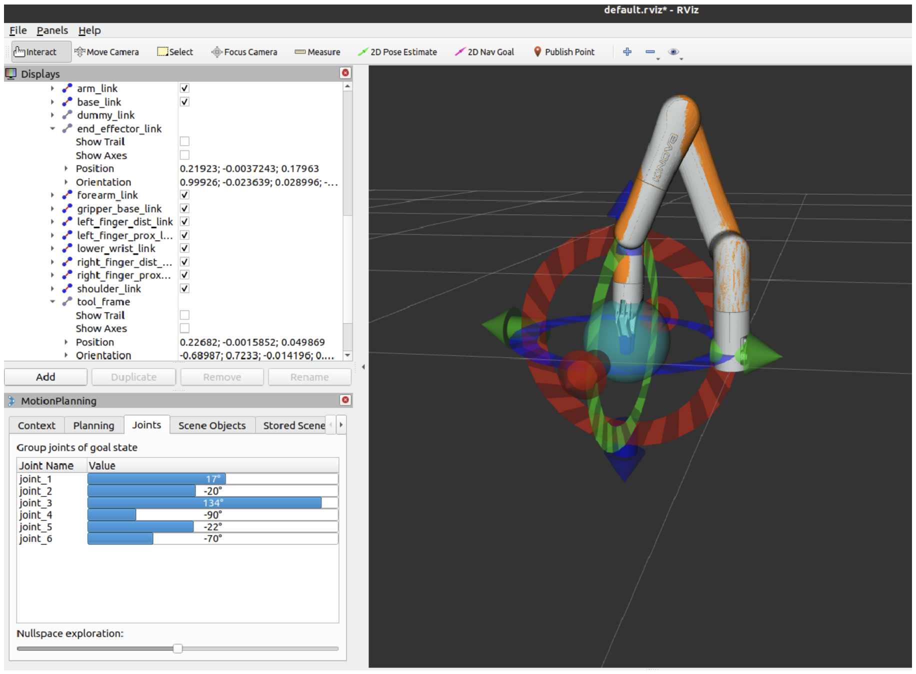
Linear	*x*	*y*	*z*
Position	0.22	0.00	0.18

(b) Orientation

Angular	*x*	*y*	*z*	*w*
Orientation	1.00	-0.02	0.04	-0.01

Table 8: Joint angles of end effector in custom state

Joint	*1*	*2*	*3*	*4*	*5*	*6*
Radian	0.30	-0.35	2.34	-1.57	-0.38	-1.22
Degree	17	-20	134	-90	-22	-70



```
arm_link                ✓
base_link               ✓
dummy_link
end_effector_link
  Show Trail
  Show Axes
  Position              0.21923; -0.0037243; 0.17963
  Orientation           0.99926; -0.023639; 0.028996; -...
forearm_link            ✓
gripper_base_link       ✓
left_finger_dist_link   ✓
left_finger_prox_l...   ✓
lower_wrist_link        ✓
right_finger_dist_...   ✓
right_finger_prox...    ✓
shoulder_link           ✓
tool_frame
  Show Trail
  Show Axes
  Position              0.22682; -0.0015852; 0.049869
  Orientation           -0.68987; 0.7233; -0.014196; 0....
```

Group joints of goal state

Joint Name	Value
joint_1	17°
joint_2	-20°
joint_3	134°
joint_4	-90°
joint_5	-22°
joint_6	-70°

Nullspace exploration:

Figure 5: Custom state of Kinova arm in RViz.

2. Explain the differences you observed when controlling the robot's position using either the interactive markers or the joint angles.

   Answer: When controlling with the interactive marker, the entire arm moves together, whereas adjusting the angle in the Joint tab moves each corresponding joint.
   Serving as the controller for the robot's end effector, the interactive marker changes its positions and orientations, which inevitably requires the synchronized movement of all the joints. This, by definition, is inverse kinematics, the process of which joint parameters are calculated given the end of a kinematic chain.
   Changing the position and orientation of each joint from the tab requires less synchronized movement. Upon hitting 'Plan & Execute' after tuning the angles, the end effector is led to the new position, which is computed with kinematic equations. This, by definition, is forward kinematics.

3. Can you change all joint angles from the RViZ window? If not, explain what degree of freedom is not available and how that impacts the poses you can reach with the robot.

   Answer: All the joints can be rotated to almost the full range of degrees for the Kinova arm from the RViz window.

3 Introducing Fetch Robot

3.1 Fetch Packages Installation

Following the instructions to install packages from from Fetch Robotics.

```
$ cd catkin_ws/src
$ git clone -b ros1 https://github.com/fetchrobotics/fetch_ros.git
$ git clone -b gazebo11 https://github.com/fetchrobotics/fetch_gazebo.git
$ git clone -b ros1 https://github.com/fetchrobotics/fetch_msgs.git
$ git clone -b ros1 https://github.com/fetchrobotics/power_msgs.git
$ git clone -b ros1 https://github.com/fetchrobotics/robot_controllers.git
$ cd ../
$ rosdep install --from-paths src --ignore-src -y
(Ignore the error: Unable to locate package ros-noetic-simple-grasping)
$ sudo apt install ros-noetic-rgbd-launch
```

Note:The branch "ros1" and "gazebo11" is developed for ROS Noetic. If you are using other ROS distributions, please git clone to corresponding branch.

3.2 Fetch Robot Manipulation

We will go through a similar procedure as the Kinova Gen3lite, familiarizing ourselves with the arm components of the Fetch robot. Execute the following commands in different terminals to spawn a Gazebo simulated Fetch robot, activate the motion capabilities and open an RViZ window

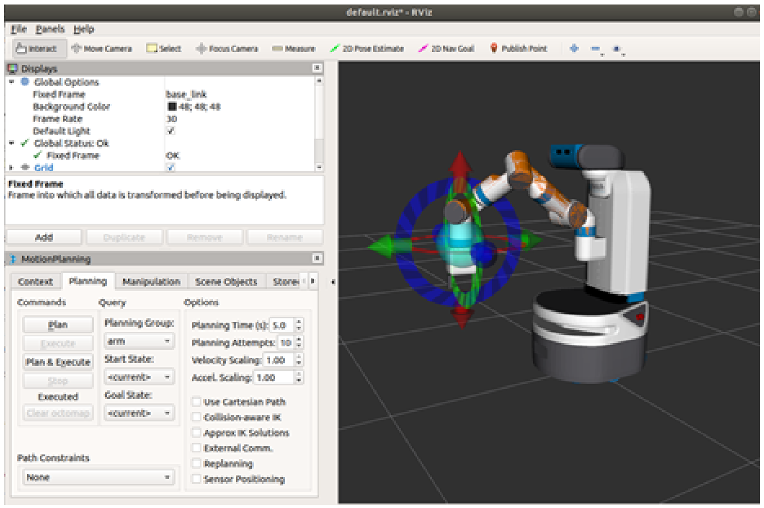
```
$ roslaunch fetch_gazebo simulation.launch
$ roslaunch fetch_moveit_config move_group.launch
$ rosrun rviz rviz
```

You will need to add the Robot Model and the MotionPlanning components in RViZ once it opens. Check the rqt_tf_tree to determine the name of the end-effector's link to run the command:

```
$ rosrun tf tf_echo /odom <END-EFFECTOR LINK>
```

You may also consider using the start of the arm chain as the reference:

```
$ rosrun tf tf_echo /<FIRST ARM JOINT> <END-EFFECTOR LINK>
```



Global Options
Fixed Frame base_link
Background Color ■ 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Ok
✓ Fixed Frame OK
Grid ✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Executed
Planning Group: arm
Start State: <current>
Goal State: <current>
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning
Path Constraints
None

Figure 6: Fetch in Rviz.

Deliverables:

1. Add a TF component in RViz and enable only the base_link, the first joint of the arm, and the end_effector_link. Disable the Motion Planning component to remove temporarily the interactive markers. Add a screenshot of the RViZ view.

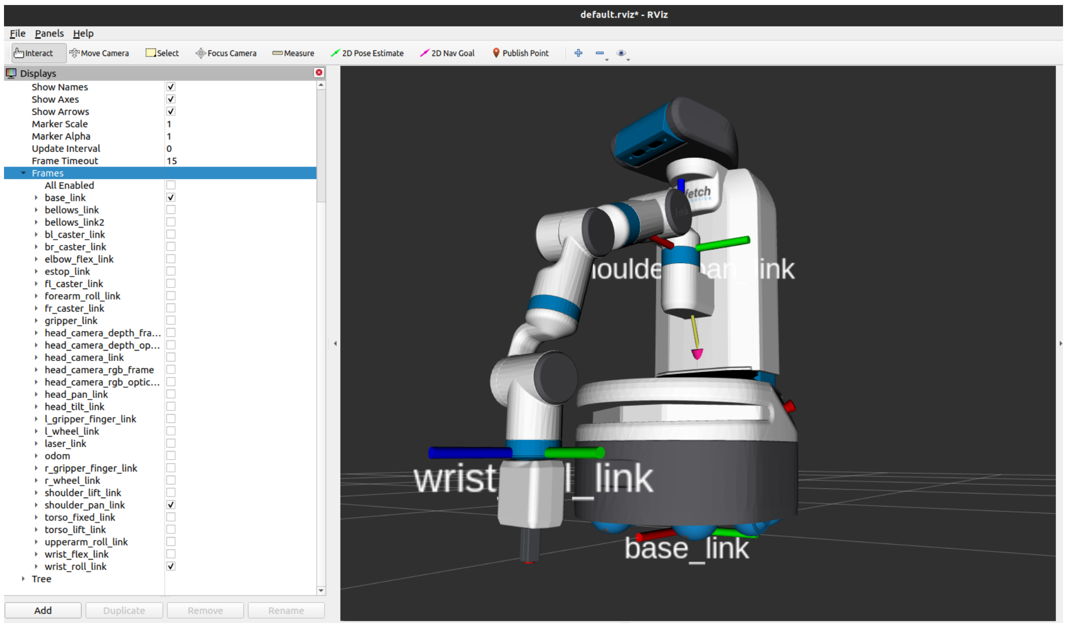
   Answer: Figure 7 shows a spawned Fetch robot in RViz with only certain TF components enabled and MotionPlanning disabled. The components requested include links to the base, the first arm joint, and the end effector. Since the definitions of these parts can vary, the following relationships between components and link names are assumed for clarification:

   ```
   <BASE LINK> = base_link
   <FIRST ARM JOINT> = shoulder_pan_link
   <END-EFFECTOR LINK> = wrist_roll_link
   ```

2. Use rqt_tf_tree to determine the kinematic chain between the two reference frames.

   Answer: According to the flowchart shown in `rqt_tf_tree`, the kinematic chain between reference frames, the first joint of arm `shoulder_pan_link` and the end effector `wrist_roll_link`, can be represented as:

   $base\_link \rightarrow torso\_lift\_link \rightarrow shoulder\_pan\_link \rightarrow shoulder\_lift\_link \rightarrow upperarm\_roll\_link \rightarrow elbow\_flex\_link \rightarrow forearm\_roll\_link \rightarrow wrist\_flex\_link \rightarrow wrist\_roll\_link \rightarrow gripper\_link$



Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Marker Alpha	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	☐
base_link	✓
bellows_link	☐
bellows_link2	☐
bl_caster_link	☐
br_caster_link	☐
elbow_flex_link	☐
estop_link	☐
fl_caster_link	☐
forearm_roll_link	☐
fr_caster_link	☐
gripper_link	☐
head_camera_depth_fra...	☐
head_camera_depth_op...	☐
head_camera_link	☐
head_camera_rgb_frame	☐
head_camera_rgb_optic...	☐
head_pan_link	☐
head_tilt_link	☐
l_gripper_finger_link	☐
l_wheel_link	☐
laser_link	☐
odom	☐
r_gripper_finger_link	☐
r_wheel_link	☐
shoulder_lift_link	☐
shoulder_pan_link	✓
torso_fixed_link	☐
torso_lift_link	☐
upperarm_roll_link	☐
wrist_flex_link	☐
wrist_roll_link	✓
Tree	

Figure 7: Enabling certain TF components in RViz.

3. Unlike the Kinova arm, the MotionPlanning for Fetch does not have preset poses for the arm.

   (a) Show a screenshot of the resulting pose when all joints in the arm are set to 0 degrees.

   Answer: Figure 8 shows an attempt to set all the joint angles to zero. However, this is the best it can be done. It was found that some joints couldn't be controlled in the Joints tab, while interactive marker couldn't independently control single joints; setting current non-zero angles to zero would also impact on other angles.

   (b) Report the joint angle values of all Fetch's arm joints on a pose that would allow Fetch to pick up an object from the floor.

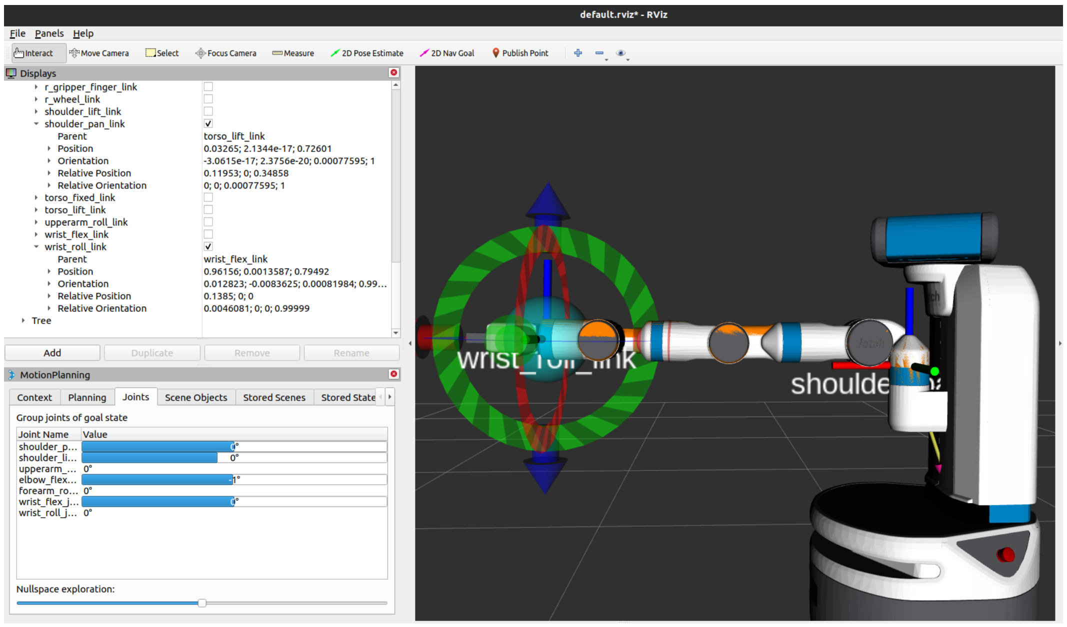
   Answer: Figure 9 shows the state, in which the Fetch robot can pick up an object from the ground. The set of joint angle values can be found in the Joint tab.

4. Explain the differences you observed when controlling the robot's position using either the interactive markers or the joint angles.

   Answer: As explained in 2.4.2, controlling with interactive markers is inverse kinematics, and with Joint tab slide bars is forward kinematics. It remains the same that moving the interactive marker will change the angle for all joints, and that most joints can be tuned independently. The difference is that changes to some joint angles can only be made with interactive markers, and will inevitably impact on other joints.

5. Can you change all joint angles from the RViZ window? If not, explain what degree of freedom is not available and how that impacts the poses you can reach with the robot.

   Answer: No, not all joint angles can be changed from the Joints tab in RViz.
   Among all the joints, `upperarm_roll_link`, `forearm_roll_link` and `wrist_roll_link`



r_gripper_finger_link	
r_wheel_link	
shoulder_lift_link	
shoulder_pan_link	✓
Parent	torso_lift_link
Position	0.03265; 2.1344e-17; 0.72601
Orientation	-3.0615e-17; 2.3756e-20; 0.00077595; 1
Relative Position	0.11953; 0; 0.34858
Relative Orientation	0; 0; 0.00077595; 1
torso_fixed_link	
torso_lift_link	
upperarm_roll_link	
wrist_flex_link	
wrist_roll_link	✓
Parent	wrist_flex_link
Position	0.96156; 0.0013587; 0.79492
Orientation	0.012823; -0.0083625; 0.00081984; 0.99...
Relative Position	0.1385; 0; 0
Relative Orientation	0.0046081; 0; 0; 0.99999
Tree	

Group joints of goal state

Joint Name	Value
shoulder_p...	0°
shoulder_li...	0°
upperarm_...	0°
elbow_flex...	-1°
forearm_ro...	0°
wrist_flex_j...	0°
wrist_roll_j...	0°

Nullspace exploration:

Figure 8: Setting joint angles to zero.

can't be tuned with the sliders, but only through interactive markers. That is, none of the roll joints can be controlled in isolation.

For the ones supporting independent manipulation, the degree range of motion is also limited. Unlike the Kinova arm, where the ranges are almost all near $[-180, 180]$, those of the Fetch robot are much more smaller. The joint degree range can be as small as $[-70, 87]$, which belongs to the `shoulder_lift_link`, preventing it from hitting its head. The pattern is that any joint parameter combination that will invite accidental collision with its own torso is prohibited.

4 Using Python to control Fetch robots

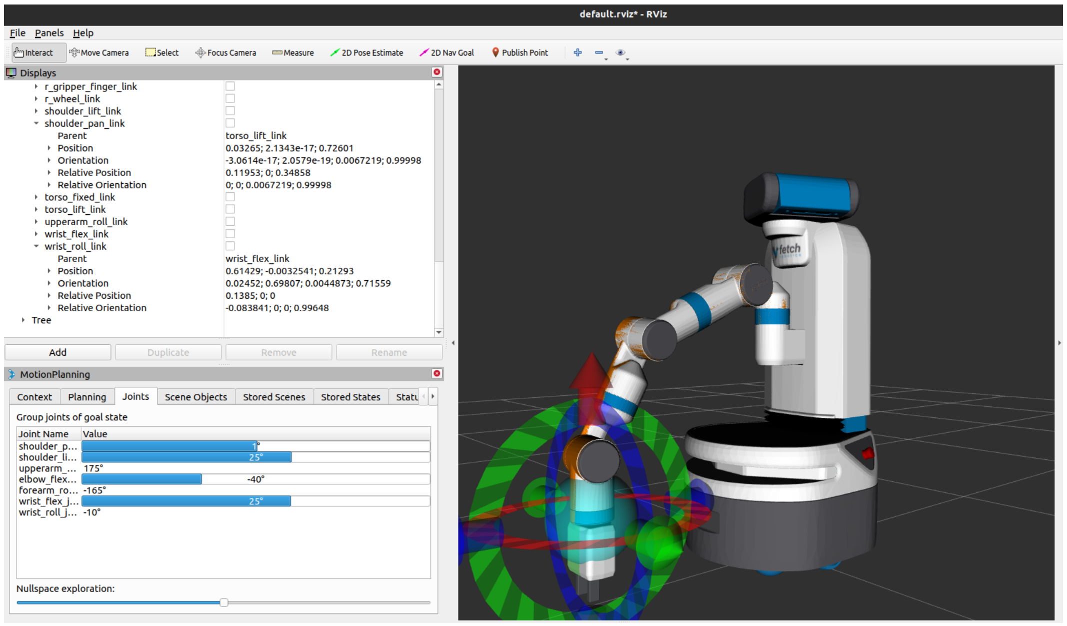
Now that we have familiarized the robot, we will create nodes that will generate motions on Fetch robot. One will move the robot arm by assigning values to joints angle, applying **forward kinematics** to reach to the target position and orientation of the end-effector in Cartesian space. Another node will apply **inverse kinematics** which calculate and change the joint angles based on the given target pose of the end-effector. You will use working code provided on canvas as a guide. You may need to modify the code to finish the deliverable.

Notes: You can visit the Fetch documentation for additional tutorials.

Deliverables:

1. Please identify which Python file corresponds to each type of robot kinematics?

   Answer: Since joint parameters are computed to move the robot to the goal state, forward



```
r_gripper_finger_link
r_wheel_link
shoulder_lift_link
shoulder_pan_link
    Parent                 torso_lift_link
    Position               0.03265; 2.1343e-17; 0.72601
    Orientation            -3.0614e-17; 2.0579e-19; 0.0067219; 0.99998
    Relative Position      0.11953; 0; 0.34858
    Relative Orientation   0; 0; 0.0067219; 0.99998
torso_fixed_link
torso_lift_link
upperarm_roll_link
wrist_flex_link
wrist_roll_link
    Parent                 wrist_flex_link
    Position               0.61429; -0.0032541; 0.21293
    Orientation            0.02452; 0.69807; 0.0044873; 0.71559
    Relative Position      0.1385; 0; 0
    Relative Orientation   -0.083841; 0; 0; 0.99648
Tree
```

Group joints of goal state

Joint Name	Value
shoulder_p...	1°
shoulder_li...	25°
upperarm_...	175°
elbow_flex...	-40°
forearm_ro...	-165°
wrist_flex_j...	25°
wrist_roll_j...	-10°

Nullspace exploration:

Figure 9: Picking up from the ground with Fetch.

kinematics is used in `simple_disco.py`. As the joint angles are calculated based on the pose of the end effector pose, `wave.py` uses inverse kinematics.

2. Capture videos of the robot executing movements using both programs, separately. Upload them to Canvas with your writeup.

   Google Drive link to Fetch robot's simple disco move

   Google Drive link to Fetch robot's wave

3. Modify the code that performs forward kinematics and move the arm to a pose with your desired input values. Compare the resulting angle values (rostopic echo /joint_states) to your inputs.

Table 9: Joint angles of Fetch torso parts in custom state

Source	*torso lift*	*shoulder pan*	*shoulder lift*	*upperarm roll*	*elbow flex*	*forearm roll*	*wrist flex*	*wrist roll*
`simple_disco.py`	0.00	0.65	-0.77	2.15	-1.17	0.73	-1.40	-0.77
`joint_states`	0.00	0.65	-0.76	2.15	-1.16	0.73	-1.40	-0.77

Answer: The joint parameters reported in `joint_states` highly agree with the ones set in `simple_disco.py`, despite some slight errors.

4. Modify the code that performs inverse kinematics and move the arm to a pose with your desired input values. Then compare the resulting pose (rosrun tf tf_echo /<link1> /<link2) to your inputs.

Table 10: Pose information of end effector in custom state

(a) Position

Source	x	y	z
`wave.py`	0.042	0.384	1.826
`tf_echo`	0.042	0.384	1.826

(b) Orientation

Source	x	y	z	w
`wave.py`	0.173	-0.693	-0.242	0.657
`tf_echo`	0.175	-0.694	-0.240	0.656

Answer: The pose information reported in `tf_echo` highly agrees with the one set in `wave.py` as well, despite some slight errors.

5. Discuss what type of robot kinematics was faster to use? Which one was easier to use?

   Answer: Forward kinematics is much more efficient in navigating the arm to the goal state, as the only thing required is the ending pose of the end effector. All the joint parameters can be calculated based on this goal, which save a ton of time.
   In terms of accuracy and safety, inverse kinematics is the one to pick. Respectively setting each joint angle dictates the shape of the arm and thus prevents accidental collisions. But controlling the arm in RViz is different from terminal, and would yield different results.

6. What were the advantages and disadvantages of using RViz or generated code to control the robot's motion? Please provide at least two thoughts.

RVIZ:

Pros:

- easier to navigate
- preview goal state
- safety warnings

Cons:

- partial joint control
- can't set goal pose

CODE:

Pros:

- full control over arm
- accurate in navigation
- easier to deploy

Cons:

- hard to visualize
- manual calculations