Panda Programming Guide

Ahmad AlAttar

March 12, 2019

Table of Content

1 Introduction

Franka Emika's Panda robot is a 7 DOF manipulator arm, shown below. The Panda Programming Guide will help you install the required packages, turn On/Off the robot, and how to program it using Python.

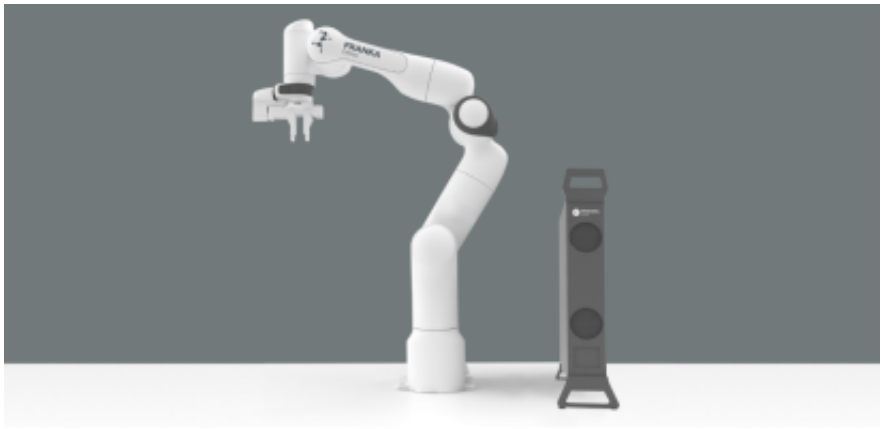

Figure 1: Franka Emika's Panda robot

2 Installation Guide

If a workspace exists with libfranka, franka_ros, and panda_moveit_config installed, **then skip the Installation Guide**.

2.1 Workspace Setup

In order to create ROS packages later on, we first need a catkin workspace. Follow these steps:

1. Go to your home directory:

   terminal: cd

2. Create a directory called Franka_ws. This will be your workspace:

   terminal: mkdir Franka_ws

3. Go inside the workspace:

   terminal: cd Franka_ws

4. Create a folder called src:

terminal: mkdir src

5. Build your workspace:

terminal: catkin_make

2.2 Installation of libfranka

The library libfranka is the C++ implementation of the client side of the Franka Control Interface (FCI). Follow these steps to install libfranka:

1. Go into the src folder of your workspace:

terminal: cd Franka_ws/src

2. Execute this command to install required dependencies:

terminal: sudo apt install build-essential cmake git libpoco-dev libeigen3-dev

3. Clone libfranka with this command:

terminal: git clone –recursive https://github.com/frankaemika/libfranka

4. Go back to the root of your workspace:

terminal: cd ..

5. Build your workspace:

terminal: catkin_make

2.3 Installation of franka_ros

The franka_ros metapackage integrates libfranka into ROS. Follow these steps to install libfranka:

1. Go into the src folder of your workspace:

terminal: cd Franka_ws/src

2. Clone libfranka with this command:

terminal: git clone –recursive https://github.com/frankaemika/franka_ros

3. Go back to the root of your workspace:

terminal: cd ..

4. Build your workspace:

terminal: catkin_make

2.4 Installation of panda_moveIt_config

The panda_moveIt_config enabled the use of MoveIt to do motion planning for the Panda robot. Follow these steps to install panda_moveIt_config:

1. Go into the src folder of your workspace:

   terminal: cd Franka_ws/src

2. Clone panda_moveIt_config with this command:

   terminal: git clone -b kinetic-devel
   https://github.com/ros-planning/panda_moveit_config.git

3. Go back to the root of your workspace:

   terminal: cd ..

4. Build your workspace:

   terminal: catkin_make

3 Turn On/Off Panda

This section details how to turn On/Off the Panda robot.

3.1 Turn On Panda

Before programming, it is important to have the Panda robot in the Ready mode. Follow these steps to do so:

1. Connect all the wires needed.
2. Turn on the controller (you will see a blinking yellow light).
3. Open a browser.
4. Go to 192.168.0.88 using the browser.
5. Unlock the joint brakes (when done, you will see white or purple light).
6. If not already, set the E-Stop to Ready mode (you will see blue light).

3.2 Turn Off Panda

After you are done, make sure to switch off the Panda following these steps. **DO NOT DIRECTLY SWITCH OFF THE CONTROLLER**.

1. In the web interface click Shut down.
2. Wait for shutdown to complete (when done, panda lights will switch off).
3. Switch off the controller.

4 Panda Gripper Control

This section shows how to control the Panda robot gripper using action servers and MoveIt.

4.1 Launch Gripper Node

To be able to control the gripper, we launch the franka_gripper node **(NOT NEEDED IF franka_control IS ALREADY LAUNCHED)**. Follow these steps:

1. Go to the workspace:

   terminal 1: cd Franka_ws

2. Launch the franka_gripper node in a terminal:

   terminal 1: roslaunch franka_gripper franka_gripper.launch robot_ip:=192.168.0.88

3. In another terminal, you can check that the node was launched:

   terminal 2: rosnode list

4.2 Gripper Action Servers

After launching the franka_gripper node, these action serves will be available to you:

1. MoveAction(width, speed)
2. GraspAction(width, epsilon_inner, epsilon_outer, speed, force)
3. HomingAction()
4. StopAction()

All of the action servers can be used in Python. To send commands to the action servers, we need action clients.

4.2.1 Gripper Homing Action

The following lines of code show how to create a simple action client for the HomingAction action server:

```
import rospy
import actionlib
from franka_gripper.msg import HomingAction, HomingGoal

if __name__ == '__main__':
    rospy.init_node('Franka_gripper_homing_action')
    client = actionlib.SimpleActionClient('/franka_gripper/homing', HomingAction)
    client.wait_for_server()
    goal = HomingGoal()
    client.send_goal(goal)
    client.wait_for_result(rospy.Duration.from_sec(5.0))
```

This action server homes the gripper and updates the maximum width given the mounted fingers.

4.2.2 Gripper Move Action

The following lines of code show how to create a simple action client for the MoveAction action server:

```
import rospy
import actionlib
from franka_gripper.msg import MoveGoal, MoveAction

if __name__ == '__main__':
    rospy.init_node('Franka_gripper_move_action')
    client = actionlib.SimpleActionClient('/franka_gripper/move', MoveAction)
    client.wait_for_server()
    goal = MoveGoal(width = 0.08, speed = 0.08)
    client.send_goal(goal)
    client.wait_for_result(rospy.Duration.from_sec(5.0))
```

This action server moves to a target width with the defined speed.

4.2.3 Gripper Stop Action

The following lines of code show how to create a simple action client for the StopAction action server:

```
import rospy
import actionlib
from franka_gripper.msg import StopAction, StopGoal

if __name__ == '__main__':
    rospy.init_node('Franka_gripper_stop_action')
    client = actionlib.SimpleActionClient('/franka_gripper/stop', StopAction)
    client.wait_for_server()
```

```
action = StopGoal()
client.send_goal(action)
client.wait_for_result(rospy.Duration.from_sec(5.0))
```

This action server aborts a running action. This can be used to stop applying forces after grasping.

4.3 Gripper Control using MoveIt

MAKE SURE EITHER franka_control NODE OR franka_gripper NODE IS LAUCNHED. Next we will need to launch MoveIt and RViz (RViz is optional).

4.3.1 Launch MoveIt

Follow these steps to launch MoveIt:

1. Go to the workspace:

   terminal 2: cd Franka_ws

2. Launch the MoveIt in a terminal:

   terminal 2: roslaunch panda_moveit_config panda_moveit.launch controller:=position

4.3.2 Launch RViz

RViz is used for visualization. Follow these steps to launch RViz (optional):

1. Go to the workspace:

   terminal 3: cd Franka_ws

2. Launch the MoveIt in a terminal:

   terminal 3: roslaunch panda_moveit_config moveit_rviz.launch

4.3.3 Setup

Follow these setup steps (in a new Python file):

1. Import the following libraries:

```
import sys
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
```

2. Initialize moveit_commander and rospy:

```
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface', anonymous=True)
```

3. Instantiate a RobotCommander object which is the outer-level interface to the robot:

```
robot = moveit_commander.RobotCommander()
```

4. Instantiate a PlanningSceneInterface object which is an interface to the world surrounding the robot:

```
scene = moveit_commander.PlanningSceneInterface()
```

5. Instantiate a MoveGroupCommander object which is an interface to one group of joints:

```
group_name = "hand"
group = moveit_commander.MoveGroupCommander(group_name)
```

6. Create a DisplayTrajectory publisher which is used to publish trajectories for RViz to visualize:

```
display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory, queue_size=20)
```

4.3.4 Gripper Control

The following lines of code uses MoveIt to open the gripper:

```
joint_goal = group.get_current_joint_values()
joint_goal[0] = 0.03
joint_goal[1] = 0.03

group.go(joint_goal, wait=True)
group.stop()
```

The following lines of code uses MoveIt to close the gripper:

```
joint_goal = group.get_current_joint_values()
joint_goal[0] = 0.00
joint_goal[1] = 0.00

group.go(joint_goal, wait=True)
group.stop()
```

5 Franka Control

In this section, we will control the Panda arm using MoveIt in Python. It is important to first launch the required files: franka_control, MoveIt, and RViz (RViz is optional).

5.1 Launch Nodes

5.1.1 Launch Control Node

The franka_control node exposes ROS services for controlling the Panda robot. Follow these steps to launch the franka_control node:

1. Go to the workspace:

   terminal 1: cd Franka_ws

2. Launch the franka_control node in a terminal:

   terminal 1: roslaunch franka_control franka_control.launch robot_ip:=192.168.0.88 load_gripper:=true

3. In another terminal, you can check that the node was launched:

   terminal 2: rosnode list

5.1.2 Launch MoveIt

Follow these steps to launch MoveIt:

1. Go to the workspace:

   terminal 2: cd Franka_ws

2. Launch the MoveIt in a terminal:

   terminal 2: roslaunch panda_moveit_config panda_moveit.launch controller:=position

5.1.3 Launch RViz

RViz is used for visualization. Follow these steps to launch RViz (optional):

1. Go to the workspace:

   terminal 3: cd Franka_ws

2. Launch the MoveIt in a terminal:

   terminal 3: roslaunch panda_moveit_config moveit_rviz.launch

5.2 Panda Control using MoveIt

This is where the robot actually move! **MAKE SURE YOU LAUCNHED franka_control, MoveIt, AND RViz (optional)**. Run your code in terminal 4.

5.2.1 Setup

Follow these setup steps (in a new Python file):

1. Import the following libraries:

```
import sys
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import pi
```

2. Initialize moveit_commander and rospy:

```
moveit_commander.roscpp_initialize(sys.argv)
rospy.init_node('move_group_python_interface', anonymous=True)
```

3. Instantiate a RobotCommander object which is the outer-level interface to the robot:

```
robot = moveit_commander.RobotCommander()
```

4. Instantiate a PlanningSceneInterface object which is an interface to the world surrounding the robot:

```
scene = moveit_commander.PlanningSceneInterface()
```

5. Instantiate a MoveGroupCommander object which is an interface to one group of joints:

```
group_name = "panda_arm"
group = moveit_commander.MoveGroupCommander(group_name)
```

6. Create a DisplayTrajectory publisher which is used to publish trajectories for RViz to visualize:

```
display_trajectory_publisher =
rospy.Publisher('/move_group/display_planned_path',
moveit_msgs.msg.DisplayTrajectory, queue_size=20)
```

5.2.2 Basic Information Print

To get some basic information, you can run the following:

```
# To print the reference frame of robot:
planning_frame = group.get_planning_frame()
print planning_frame

# To print the name of the end-effector link for this group:
eef_link = group.get_end_effector_link()
print eef_link

# To print a list of all the groups in the robot:
group_names = robot.get_group_names()
print group_names

# To print the entire state of the robot:
robot_state = robot.get_current_state()
print robot_state
```

5.2.3 Simple Move

To move the robot, run the following piece of code:

```
joint_goal = group.get_current_joint_values()
joint_goal[0] = 0
joint_goal[1] = -pi/4
joint_goal[2] = 0
joint_goal[3] = -pi/2
joint_goal[4] = 0
joint_goal[5] = pi/3
joint_goal[6] = 0

group.go(joint_goal, wait=True)
group.stop() # makes sure no residual movements left
```

5.2.4 Planning to Pose Goal

To plan a simple pose and move the robot, run the following piece of code:

```
pose_goal = geometry_msgs.msg.Pose()
pose_goal.orientation.w = 1.0
pose_goal.position.x = 0.4
pose_goal.position.y = 0.1
pose_goal.position.z = 0.4
group.set_pose_target(pose_goal)

plan = group.go(wait=True)
```

```
group.stop()
group.clear_pose_targets() # clear targets after planning
```