7) Writing a basic Takktile script

The next tutorial is 8) Getting under the hood

In previous tutorials we used the command line to understand how the hand works. Here we'll go through examples of controlling the hand with python. The easiest way to see all the functions available to you is to check reflex_hand.py, and reflex_takktile_hand.py, where the functions are written.

Example code has been written for you. Inside of the reflex package you will find takktile_example_code.py. You're encouraged to run the code and go through it yourself, commenting out lines or changing lines to see what affect it has on the hand behavior. In this tutorial we will talk about a few of the important moments, but for full understanding you should explore the example script.

Useful imports

To command the hand a number of ROS messages are used. Note that most of the imports come from reflex_msgs.msg, which are the messages defined in the reflex_msgs package.

```
from std_srvs.srv import Empty

from reflex_msgs.msg import Command
from reflex_msgs.msg import PoseCommand
from reflex_msgs.msg import VelocityCommand
from reflex_msgs.msg import Hand
from reflex_msgs.msg import FingerPressure
from reflex_msgs.srv import SetTactileThreshold, SetTactileThresholdRequest
```

Also important, if you want to set the tactile thresholds using the /set_tactile_threshold service, you'll have to import and use the SetTactileThresholdRequest class. When calling services ROS requires the addition of Request to the service type.

Setting up publishers, services, and subscribers

To command your hand you'll likely want some combination of the calls below. In order to automatically calibrate the fingers or the tactile sensors, you'll need to call the ServiceProxy functions.

In order to send position/velocity commands to the hand, you'll want to make publishers that publish to the appropriate topic, as shown here.

Finally, if you want to monitor the state of the hand you'll want to make a subscriber to the /hand_state topic. That will lets you keep track of the joint positions, whether the fingers are in contact, etc.

```
# Services can automatically call hand calibration
calibrate_fingers = rospy.ServiceProxy('/reflex_takktile/calibrate_fingers', Empty)
calibrate_tactile = rospy.ServiceProxy('/reflex_takktile/calibrate_tactile', Empty)

# Services can set tactile thresholds and enable tactile stops
enable_tactile_stops = rospy.ServiceProxy('/reflex_takktile/enable_tactile_stops',
disable_tactile_stops = rospy.ServiceProxy('/reflex_takktile/disable_tactile_stops'
set_tactile_threshold = rospy.ServiceProxy('/reflex_takktile/set_tactile_threshold'

# This collection of publishers can be used to command the hand
command_pub = rospy.Publisher('/reflex_takktile/command', Command, queue_size=1)
pos_pub = rospy.Publisher('/reflex_takktile/command_position', PoseCommand, queue_s
vel_pub = rospy.Publisher('/reflex_takktile/command_velocity', VelocityCommand, que

# Constantly capture the current hand state
rospy.Subscriber('/reflex_takktile/hand_state', Hand, hand_state_cb)
```

If you want more information about how to make publishers and subscribers, try checking out the introductory ROS tutorials on the subject.

Publishing commands

There are many examples of publishing in the example code, but here is a basic illustration. The first thing to do is create a message of the right type, which we do with the VelocityCommand() call. The VelocityCommand can be populated by setting each of its elements - f1, f2, f3, and preshape. If you leave out any elements they will be set to 0.0, which could be desired. Finally, the command is published with vel_pub.publish().

```
vel_pub.publish(VelocityCommand(f1=-5.0, f2=-5.0, f3=-5.0, preshape=0.0))
```

Setting tactile thresholds

If you want to adjust the tactile thresholds it requires a little work, shown here. Each finger with nine sensors needs to be created as a FingerPressure object, then the SetTactileThresholdRequest needs to be created with a three-element list of FingerPressure messages.

```
f1 = FingerPressure([10, 10, 10, 10, 10, 10, 10, 10, 1000])
f2 = FingerPressure([15, 15, 15, 15, 15, 15, 15, 15, 1000])
f3 = FingerPressure([20, 20, 20, 20, 20, 20, 20, 20, 1000])
threshold = SetTactileThresholdRequest([f1, f2, f3])
set_tactile_threshold(threshold)
```

Note that the last sensor on each finger is set to a threshold of 1000. This is an example of how to disable a sensor. The fingertip sensor will never register contact after that threshold is set, until it is reset, which might be useful if you wanted to do a grasp that ignored the fingertip contact.

Keep on going to 8) Getting under the hood