ECE470: Lab 2 - The Tower of Hanoi with ROS

Name of Student: Yuhang Chen
NetID: yuhang.17
Lab Partner: Yuxin Zhao
TA: Zhefan Lin
Section: Wednesday 3PM
Submitted on Apr. 14, 2021

1 Introduction

In this lab, the main task is to control the UR3e robot using the Robot Operating System (ROS) and the Python Programming language to deal with the Tower of Hanoi puzzle.

The puzzle starts with three blocks in a neat stack in ascending order of size on the one location on the table, the smallest at the top. And the blocks from largest size to smallest size are named from block 1 to block 3.

The goal of the puzzle is to move the entire stack to another location on the table, meanwhile obeying the following simple rules:

1) Only one block can be moved at a time.
2) Each move consists of taking the upper block from one of the stacks and placing it on top of another stack or on an empty location.
3) No larger block may be placed on top of a smaller block.

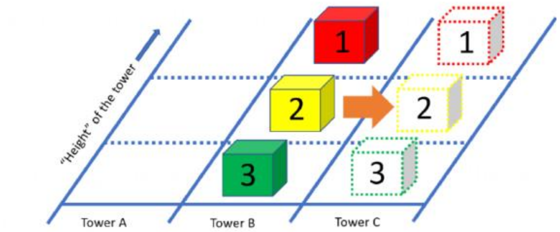

Figure 1.1: Example start and finish tower locations.

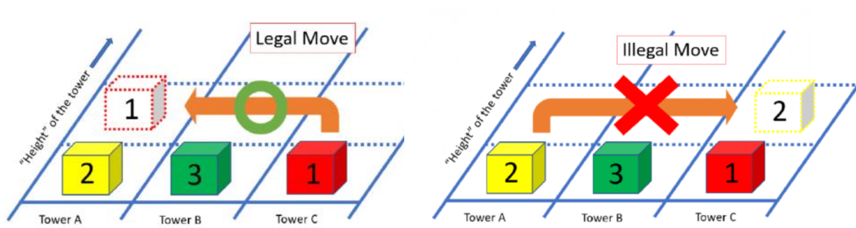

Figure 1.2: Examples of a legal and an illegal move.

2 Procedure

The Main Focus on this lab

In this lab, the task for our team to do mainly focus on the ROS and implementing feedback.

Key steps of using ROS shows as the following:

Step 1. Build up own workspace on the host.

To begin with, we need to build up our own workspace on the computer. We build an empty file folder and copy the original scripts provided by the lab materials into this file folder by using Linux commands.

Step 2. Make program for dealing with the Tower of Hanoi puzzle using ROS.

The lab materials have already provided the overall framework and some parts of the python codes for this lab. Lab2_header.py file provides with the feedback information that we can get from UR3e robot. All we need to do is to modify the codes in lab2_exec.py and implement the interactive operations to achieve the program to deal with the Tower of Hanoi puzzle. To be more specific, we need to complete three parts of the codes:

Part 1. The definition of the Waypoints Matrix Q we make for the Tower of Hanoi

By the definition in the lab2_manual, Q is a list that stores all the necessary waypoints for the robot to pick and place the blocks in order to solve the Tower of Hanoi. In each entry of Q[tower index][block height][above block/on block], there are six angles in radians that correspond to the arm's six joint angles.

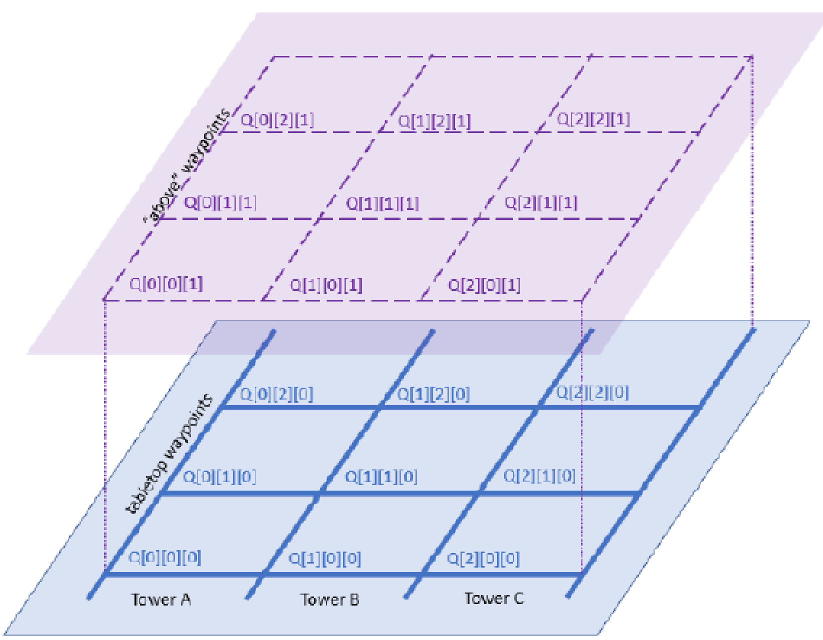

Figure 2.1 Waypoint Matrix Q

Code snippets for Q:

```
############### Your Code Start Here ###############

"""
TODO: Definition of position of our tower in Q
"""
Q02 = np.radians([-102.07, -119.49, -85.66, -47.94, 89.93, 127.06])
Q01 = np.radians([-105.23, -122.23, -86.10, -39.89, 90.60, 127.06])
Q00 = np.radians([-104.80, -126.30, -85.49, -37.80, 89.12, 127.06])
Q12 = np.radians([-107.26, -117.26, -72.46, -43.76, 89.98, 127.06])
Q11 = np.radians([-107.45, -121.05, -72.42, -41.03, 89.42, 127.06])
Q10 = np.radians([-109.72, -124.51, -72.36, -34.55, 89.72, 127.06])
Q22 = np.radians([-106.58, -117.71, -59.23, -43.72, 89.05, 127.06])
Q21 = np.radians([-105.04, -121.10, -59.40, -44.50, 89.03, 127.06])
Q20 = np.radians([-107.26, -124.81, -59.45, -37.83, 89.09, 127.06])
Q = [[Q00,Q01,Q02],[Q10,Q11,Q12],[Q20,Q21,Q22]]


################ Your Code End Here ################
```

Figure 2.2 Code Snippets for Q

The particular value for each waypoint of Q is obtained by the control panel of the UR3e robot. We record six angles in radians in each position of Q and enter it in the program.

Part 2. Three interactive subroutine functions between the host and the machine

1) Code snippets for the subroutine function **gripper_input_callback**:

```
############## Your Code Start Here ##############

"""
TODO: define a ROS topic callback funtion for getting the state of suction cup
Whenever /ur_hardware_interface/io_states publishes info this callback function is called.
"""
def gripper_input_callback(msg):

    global current_io_0
    current_io_0 = msg.digital_in_states[0].state

############### Your Code End Here ###############
```

Figure 2.3 Code snippets for the subroutine function gripper_input_callback

The function gripper_input_callback do the task to get the state of suction cup. If the suction cup is on, it will return value 1 to the global parameter current_io_0. Otherwise, it will return value 0 to the global parameter current_io_0.

2) Code snippets for the subroutine function **gripper**:

```
############### Your Code Start Here ###############

def gripper(pub_setio, io_0):

    io = Digital()
    io.pin = 0
    io.state = io_0
    pub_setio.publish(io)


############### Your Code End Here ###############
```

Figure 2.4 Code snippets for the subroutine function gripper

The function gripper can publish the message io_o (the state of the suction cup) to the UR3e robot using pub_setio.

3) Code snippets for the function **move_block**:

```
############## Your Code Start Here ##############
"""
TODO: function to move block from start to end
"""
### Hint: Use the Q array to map out your towers by location and "height".

def move_block(pub_setjoint, pub_setio, start_loc, start_height, end_loc, end_height):
    global Q

    start= Q[int(start_loc)-1][start_height]
    end = Q[int(end_loc)-1][end_height]

    move_arm(pub_setjoint, home)
    move_arm(pub_setjoint, start)
    time.sleep(0.5)
    gripper(pub_setio,suction_on)
    move_arm(pub_setjoint, home)
    move_arm(pub_setjoint, end)
    time.sleep(0.5)
    gripper(pub_setio,suction_off)
    time.sleep(1)

############### Your Code End Here ###############
```

Figure 2.5 Code snippets for the function move_block

The function move_block take the start position (location+height) and the end position (location+height) of the block of one moving operation in the procedure. And we achieve this function by using the function move_arm and gripper that have already finished before.

Part 3. The main function to deal with the Tower of Hanoi puzzle

Code snippets for the main function:

```
    # Initialize ROS node
    rospy.init_node('lab2node')

    # Initialize publisher for ur3e_driver_ece470/setjoint with buffer size of 10

    pub_setjoint = rospy.Publisher('ur3e_driver_ece470/setjoint',JointTrajectory,
queue_size=10)

    ############## Your Code Start Here ##############
    # TODO: define a ROS publisher for /ur3e_driver_ece470/setio message

    pub_setio = rospy.Publisher('/ur3e_driver_ece470/setio',Digital,queue_size=10
)

    ############### Your Code End Here ###############


    # Initialize subscriber to /joint_states and callback fuction
    # each time data is published
    sub_position = rospy.Subscriber('/joint_states', JointState, position_callbac
k)

    ############## Your Code Start Here ##############
    # TODO: define a ROS subscriber for /ur_hardware_interface/io_states message
and corresponding callback function

    sub_setio = rospy.Subscriber('/ur_hardware_interface/io_states', IOStates, gr
ipper_input_callback)

    ############### Your Code End Here ###############
```

Figure 2.7 code snippets for the initialization step of the main function

In the part of initialization, we initialize the publisher of setio and setjoint, the subscriber of setio and setjoint by using the scripts provided in the ur_hardware_interface.

```
    ############## Your Code Start Here ##############
    # TODO: modify the code below so that program can get user input

    input_done = 0

    while(input_done == 0):
        input_string1 = raw_input("Enter the start position <Either 1 2 3 or 0 to
 quit> ")
        input_string2 = raw_input("Enter the end position <Either 1 2 3 or 0 to q
uit> ")
        print("Please Confirm that the following information:")
        print("Your entered start positioon is " + input_string2 + ".\n")
        print("You entered end position" + input_string1 + ".\n")
        if(int(input_string1) == int(input_string2)):
            print("The start point and destination point are same, quitting...")
            sys.exit()

        elif (int(input_string1) == 0):
            print("Quitting... ")
            sys.exit()
        else:
            #print("Please just enter the character 1 2 3 or 0 to quit \n\n")
            input_done = 1


    ############### Your Code End Here ###############
```

Figure 2.8 Code snippets for getting the input

To get the input, we use the raw_input function. Thus, the program can know the start location of the Tower of Hanoi and the final end location we wish it to arrive.

```
    start = input_string1
    end = input_string2
    median = 0
    if (start == '1' and end == '2'):
        median = '3'
    elif (start == '1' and end == '3'):
        median= '2'
    elif (start == '2' and end == '1'):
        median = '3'
    elif (start == '2' and end == '3'):
        median = '1'
    elif (start == '3' and end== '1'):
        median = '2'
    elif (start== '3' and end == '2'):
```

```
        median = '1'

    move_arm(pub_setjoint, home)
    move_block(pub_setjoint, pub_setio, start, 2, end,   0)
    move_block(pub_setjoint, pub_setio, start, 1, median,   0)
    move_block(pub_setjoint, pub_setio, end, 0, median,   1)
    move_block(pub_setjoint, pub_setio, start, 0, end,   0)
    move_block(pub_setjoint, pub_setio, median , 1, start,   0)
    move_block(pub_setjoint, pub_setio, median, 0, end   1)
    move_block(pub_setjoint, pub_setio, start 0, end,   2)
```
Figure 2.9 Code snippets for the moving steps of the procedure

We calculate the median location for the moving steps. And then use the subroutine function move_block to move the blocks step by step until all of them reach the supposed destination that they need to go.

Step 3. Upload the program into the UR3e robot and run it on the machine.

To achieve this, we need to make Connection between the host (the computer) and the machine (the UR3e robot). And we do this by using the ROS command to run our lab2node.

In terminal one, we run roslaunch ur_robot_driver ur3e_bringup.launch + the machine ip of the UR3e robot.

In terminal two, we source the code of devel and make lab2_exec.py to be executable by chmod +x lab2_exec.py.

In terminal three, we run the ur3e_driver_ece470.

In terminal four, we run lab2_exec.py in the lab2pkg_py folder.

By doing these steps the code is uploaded into the UR3e robot and the host terminal window 4 will ask us to enter the input to make the program to continue to run in the machine.

3 Question and Conclusion

3.1 What is ROS and How does it work?

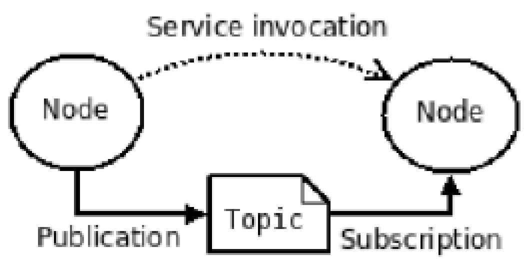

Figure 3.1 ROS System Working Procedure

In this lab, we focus on the four part of ROS: Nodes, Master, Topic and Message.

The master of ROS enables Nodes in ROS to locate one another. And Nodes publish and subscribe messages via Topic. And it can help the user to achieve the interact between the hardware (UR3e robot) and the host (the computer).

3.2 How did you use the ROS commands to complete your task?

In this lab, we use the “roslaunch” command to make connection between the host and the machine. And we use “rostopic info” to double check tht ur3e_driver_ece470/setjoint is a subscriber and “joint_states” is a publisher. Also we run “rosmsg list” to find the messages.

3.3 How did you implement feedback?

We import rospy in the lab2_exec.py to implement feedback. We define topics (i.e. gripper_input_callback), nodes (lab2node), publisher and subscriber to achieve the operations. By using rospy we succeed to interact between the host and the robot by getting the state of each part we need to know and publish the message to the machine to do each step of operation.

4 Reference

[1] *Tower of Hanoi-Wikipedia* https://en.wikipedia.org/wiki/Tower_of_Hanoi

[2] *ECE470 Lab Manual*

[3] ROS https://wiki.ros.org/

[4] Ubuntu https://wiki.ros.org/ROS/Installation

[5] ROS Tutorial https://wiki.ros.org/ROS/Tutorials

5 Appendix

All Codes for lab2_exec.py:

```
#!/usr/bin/env python

'''
We get inspirations of Tower of Hanoi algorithm from the website below.
This is also on the lab manual.
Source: https://www.cut-the-knot.org/recurrence/hanoi.shtml
'''

import copy
import time
import rospy
import actionlib
import numpy as np
from lab2_header import *

# 20Hz
SPIN_RATE = 30

# UR3 home location
home = np.radians([-94.30, -91.40, -75.12, -83.49, 91.96, 127.02])

# UR3 current position, using home position for initialization
current_position = copy.deepcopy(home)

thetas = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

digital_in_0 = 0
analog_in_0 = 0

suction_on = True
suction_off = False

current_io_0 = False
current_position_set = False

############## Your Code Start Here ##############
```

```python
"""
TODO: Definition of position of our tower in Q
"""
Q02 = np.radians([-102.07, -119.49, -85.66, -47.94, 89.93, 127.06])
Q01 = np.radians([-105.23, -122.23, -86.10, -39.89, 90.60, 127.06])
Q00 = np.radians([-104.80, -126.30, -85.49, -37.80, 89.12, 127.06])
Q12 = np.radians([-107.26, -117.26, -72.46, -43.76, 89.98, 127.06])
Q11 = np.radians([-107.45, -121.05, -72.42, -41.03, 89.42, 127.06])
Q10 = np.radians([-109.72, -124.51, -72.36, -34.55, 89.72, 127.06])
Q22 = np.radians([-106.58, -117.71, -59.23, -43.72, 89.05, 127.06])
Q21 = np.radians([-105.04, -121.10, -59.40, -44.50, 89.03, 127.06])
Q20 = np.radians([-107.26, -124.81, -59.45, -37.83, 89.09, 127.06])
Q = [[Q00,Q01,Q02],[Q10,Q11,Q12],[Q20,Q21,Q22]]


############### Your Code End Here ###############





############## Your Code Start Here ##############

"""
TODO: define a ROS topic callback funtion for getting the state of suction cup
Whenever /ur_hardware_interface/io_states publishes info this callback function i
s called.
"""
def gripper_input_callback(msg):

    global current_io_0
    current_io_0 = msg.digital_in_states[0].state

############### Your Code End Here ###############


"""
Whenever ur3/position publishes info, this callback function is called.
"""
def position_callback(msg):

    global thetas
    global current_position
    global current_position_set
```

```
    thetas[0] = msg.position[0]
    thetas[1] = msg.position[1]
    thetas[2] = msg.position[2]
    thetas[3] = msg.position[3]
    thetas[4] = msg.position[4]
    thetas[5] = msg.position[5]

    current_position[0] = thetas[0]
    current_position[1] = thetas[1]
    current_position[2] = thetas[2]
    current_position[3] = thetas[3]
    current_position[4] = thetas[4]
    current_position[5] = thetas[5]

    current_position_set = True


############## Your Code Start Here ##############

def gripper(pub_setio, io_0):

    io = Digital()
    io.pin = 0
    io.state = io_0
    pub_setio.publish(io)


############### Your Code End Here ###############

def move_arm(pub_setjoint, dest):
    msg = JointTrajectory()
    msg.joint_names = ["elbow_joint", "shoulder_lift_joint", "shoulder_pan_joint"
,"wrist_1_joint", "wrist_2_joint", "wrist_3_joint"]
    point = JointTrajectoryPoint()
    point.positions = dest
    point.time_from_start = rospy.Duration(2)
    msg.points.append(point)
    pub_setjoint.publish(msg)
    time.sleep(2.5)




############## Your Code Start Here ##############
"""
```

```
TODO: function to move block from start to end
"""
### Hint: Use the Q array to map out your towers by location and "height".

def move_block(pub_setjoint, pub_setio, start_loc, start_height, end_loc, end_hei
ght):
    global Q

    start= Q[int(start_loc)-1][start_height]
    end = Q[int(end_loc)-1][end_height]

    move_arm(pub_setjoint, home)
    move_arm(pub_setjoint, start)
    time.sleep(0.5)
    gripper(pub_setio,suction_on)
    move_arm(pub_setjoint, home)
    move_arm(pub_setjoint, end)
    time.sleep(0.5)
    gripper(pub_setio,suction_off)
    time.sleep(1)

############### Your Code End Here ###############


def main():

    global home
    global Q
    global SPIN_RATE

    # Definition of our tower

    # 2D layers (top view)

    # Layer (Above blocks)
    # | Q[0][2][1] Q[1][2][1] Q[2][2][1] |   Above third block
    # | Q[0][1][1] Q[1][1][1] Q[2][1][1] |   Above point of second block
    # | Q[0][0][1] Q[1][0][1] Q[2][0][1] |   Above point of bottom block

    # Layer (Gripping blocks)
    # | Q[0][2][0] Q[1][2][0] Q[2][2][0] |   Contact point of third block
    # | Q[0][1][0] Q[1][1][0] Q[2][1][0] |   Contact point of second block
    # | Q[0][0][0] Q[1][0][0] Q[2][0][0] |   Contact point of bottom block

    # First index - From left to right position A, B, C
```

```
    # Second index - From "bottom" to "top" position 1, 2, 3
    # Third index - From gripper contact point to "in the air" point

    # How the arm will move (Suggestions)
    # 1. Go to the "above (start) block" position from its base position
    # 2. Drop to the "contact (start) block" position
    # 3. Rise back to the "above (start) block" position
    # 4. Move to the destination "above (end) block" position
    # 5. Drop to the corresponding "contact (end) block" position
    # 6. Rise back to the "above (end) block" position

    # Initialize ROS node
    rospy.init_node('lab2node')

    # Initialize publisher for ur3e_driver_ece470/setjoint with buffer size of 10

    pub_setjoint = rospy.Publisher('ur3e_driver_ece470/setjoint',JointTrajectory,
queue_size=10)

    ############## Your Code Start Here ##############
    # TODO: define a ROS publisher for /ur3e_driver_ece470/setio message

    pub_setio = rospy.Publisher('/ur3e_driver_ece470/setio',Digital,queue_size=10
)

    ############### Your Code End Here ###############


    # Initialize subscriber to /joint_states and callback fuction
    # each time data is published
    sub_position = rospy.Subscriber('/joint_states', JointState, position_callbac
k)

    ############## Your Code Start Here ##############
    # TODO: define a ROS subscriber for /ur_hardware_interface/io_states message
and corresponding callback function

    sub_setio = rospy.Subscriber('/ur_hardware_interface/io_states', IOStates, gr
ipper_input_callback)

    ############### Your Code End Here ###############


    ############## Your Code Start Here ##############
    # TODO: modify the code below so that program can get user input
```

```
    input_done = 0

    while(input_done == 0):
        input_string1 = raw_input("Enter the start position <Either 1 2 3 or 0 to
 quit> ")
        input_string2 = raw_input("Enter the end position <Either 1 2 3 or 0 to q
uit> ")
        print("Please Confirm that the following information:")
        print("Your entered start positioon is " + input_string2 + ".\n")
        print("You entered end position" + input_string1 + ".\n")
        if(int(input_string1) == int(input_string2)):
            print("The start point and destination point are same, quitting...")
            sys.exit()

        elif (int(input_string1) == 0):
            print("Quitting... ")
            sys.exit()
        else:
            #print("Please just enter the character 1 2 3 or 0 to quit \n\n")
            input_done = 1


    ############### Your Code End Here ###############

    # Check if ROS is ready for operation
    while(rospy.is_shutdown()):
        print("ROS is shutdown!")

    rospy.loginfo("Sending Goals ...")

    loop_rate = rospy.Rate(SPIN_RATE)

    ############## Your Code Start Here ##############
    # TODO: modify the code so that UR3 can move tower accordingly from user inpu
t


    start = input_string1
    end = input_string2
    median = 0
    if (start == '1' and end == '2'):
        median = '3'
    elif (start == '1' and end == '3'):
        median= '2'
```

```
    elif (start == '2' and end == '1'):
        median = '3'
    elif (start == '2' and end == '3'):
        median = '1'
    elif (start == '3' and end== '1'):
        median = '2'
    elif (start== '3' and end == '2'):
        median = '1'

    move_arm(pub_setjoint, home)
    move_block(pub_setjoint, pub_setio, start, 2, end,   0)
    move_block(pub_setjoint, pub_setio, start, 1, median,   0)
    move_block(pub_setjoint, pub_setio, end, 0, median,   1)
    move_block(pub_setjoint, pub_setio, start, 0, end,   0)
    move_block(pub_setjoint, pub_setio, median , 1, start,   0)
    move_block(pub_setjoint, pub_setio, median, 0, end   1)
    move_block(pub_setjoint, pub_setio, start 0, end,   2)


    ############### Your Code End Here ###############



if __name__ == '__main__':

    try:
        main()
    # When Ctrl+C is executed, it catches the exception
    except rospy.ROSInterruptException:
        pass
```