1.2 LAB 1.5 Week Two, The Tower of Hanoi using the Teach Pendant

1.2.1 Important

Read the entire lab before starting.

1.2.2 Objectives

This lab is numbered 1.5 because it continues the programming you learned in Lab 1 but also prepares you for Lab 2. In Lab 2 and forward you will be using the *Robot Operating System* (ROS) environment to program the UR3. For this lab you will continue to program the UR3 using its Teach Pendant but perform a similar task that will be required in Lab 2, solving a three block Tower of Hanoi puzzle. In this lab, you will:

- Move three stacked blocks from one position to another position using the rules specified for the Tower of Hanoi puzzle. Blocks should be aligned on top of each other.
- Use high level "**Move**" commands to move the UR3's Tool Center Point in linear and circular motions
- Time permitting play with other functionality of the teach pendant.

1.2.3 References

- UR3 Owner's Manual:

  `https://www.universal-robots.com/download/manuals-cb-series/user/ur3/315/user-manual-ur3-cb-series-sw315-english-international-en/`

- UR3 Software Manual:

  `https://www.universal-robots.com/download/manuals-cb-series/software/314/software-manual-cb-series-sw314-english-us-en-us/`

- Universal Robots Academy

  `https://academy.universal-robots.com/free-e-learning/e-series-e-learning/e-series-core-track/`

- Since this is a robotics lab and not a course in computer science or discrete math, feel free to Google for solutions to the Tower of Hanoi problem.[1] You are **NOT** required to implement a recursive solution.

[1] `http://www.cut-the-knot.org/recurrence/hanoi.shtml` (an active site, as of this writing.)

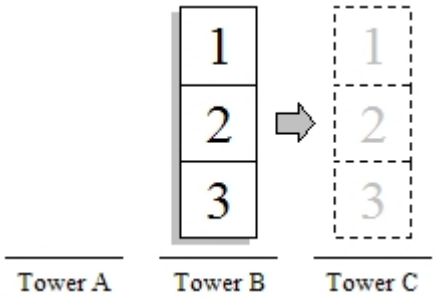

Figure 1.1: Example start and finish tower locations.

1.2.4 Pre-Lab

Read in more detail the UR3 Software Manual chapters 13 and 14. Additionally if for some reason you have not completed the training videos, go through the training videos found at Universal Robots website `https://academy.universal-robots.com/free-e-learning/e-series-e-learning/e-series-core-track/`. These training sessions get into some areas that we will not be using in this class (for example you will not be changing safety settings), but go through all of the assignments as they will help you get familiar with the UR3 and its teach pendant. You also may want to reference these sessions when you are in lab.

1.2.5 Task

The goal is to move a "tower" of three blocks from one of three locations on the table to another. An example is shown in Figure 1.1. The blocks are numbered with block 1 on the top and block 3 on the bottom. When moving the stack, two rules must be obeyed:

1. Blocks may touch the table in only three locations (the three "towers").
2. You may not place a block on top of a lower-numbered block, as illustrated in Figure 1.2.

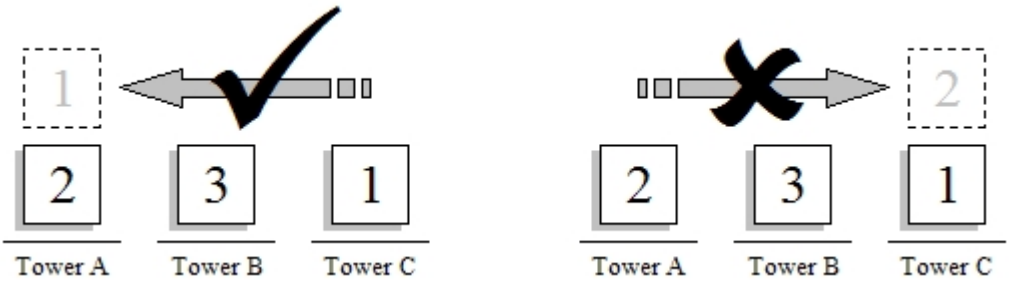

Figure 1.2: Examples of a legal and an illegal move.

1.2.6 Procedure

1. Choose the three spots on the robot's table where blocks can be placed when solving the Tower of Hanoi problem.
2. Use the provided colored tape to mark the three possible tower bases. You should initial your markers so you can distinguish your tower bases from the ones used by teams in other lab sections.
3. Choose a starting position and ending position for the tower of three blocks. Future note: In Lab 2 the user will enter the start and stop positions.
4. Using the Teach Pendant create a program that solves the Tower of Hanoi problem. Instead of using **MoveJ** moves like in Lab 1, experiment with using **MoveL** and **MoveP** moves. **MoveL** moves the Tool Center Point (TCP) along a straight line, and **MoveP** is a process move that keeps the TCP moving at a constant speed and allows you to move along circular arcs. Reference these three "How To" articles from Universal Robots on creating circular arcs:
   - `https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/circle-using-movec-16270/`
   - `https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/circular-path-using-movepmovec-15668/`
   - `https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/circle-with-variable-radius-15367/`
5. Your program must have at least one obvious linear move and one obvious circular move that completely encircles one of the block positions.

1.2.7 Demo

Show your TA the program you created.

1.2.8 Grading

- 2.5 points, attendance.
- 47.5 points, successful demonstration.

Appendix A

ROS Programming with Python

A.1 Overview

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

- The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.

- For more details about ROS: `http://wiki.ros.org/`
- How to install on your own Ubuntu: `http://wiki.ros.org/ROS/Installation`
- For detailed tutorials: `http://wiki.ros.org/ROS/Tutorials`

A.2 ROS Concepts

The basic concepts of ROS are nodes, Master, messages, topics, Parameter Server, services, and bags. However, in this course, we will only be encountering

the first four.

- `Nodes` "programs" or "processes" in ROS that perform computation. For example, one node controls a laser range-finder, one node controls he wheel motors, one node performs localization ...
- `Master` Enable nodes to locate one another, provides parameter server, tracks publishers and subscribers to topics, services. In order to start ROS, open a terminal and type:

  ```
  $ roscore
  ```

  roscore can also be started automatically when using roslaunch in terminal, for example:

  ```
  $ roslaunch <package name> <launch file name>.launch
  # the launch file for all our labs:
  $ roslaunch ur3_driver ur3_driver.launch
  ```

- `Messages` Nodes communicate with each other via messages. A message is simply a data structure, comprising typed fields.
- `Topics` Each node publish/subscribe message topics via send/receive messages. A node sends out a message by publishing it to a given topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics.In general, publishers and subscribers are not aware of each others' existence.

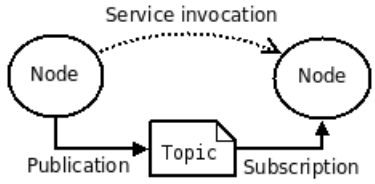

Figure A.1: source: `http://wiki.ros.org/ROS/Concepts`

A.3 Before we start..

Here are some useful Linux/ROS commands

- The command "ls" stands for (List Directory Contents), List the contents of the folder, be it file or folder, from which it runs.

  ```
  $ ls
  ```

- The "mkdir" (Make directory) command create a new directory with name path. However is the directory already exists, it will return an error message "cannot create folder, folder already exists".

  ```
  $ mkdir <new_directory_name>
  ```

- The command "pwd" (print working directory), prints the current working directory with full path name from terminal

  ```
  $ pwd
  ```

- The frequently used "cd" command stands for change directory.

  ```
  $ cd /home/user/Desktop
  ```

  return to previous directory

  ```
  $ cd ..
  ```

  Change to home directory

  ```
  $ cd ~
  ```

- The hot key "ctrl+c" in command line **terminates** current running executable. If "ctrl+c" does not work, closing your terminal as that will also end the running Python program. **DO NOT USE "ctrl+z" as it can leave some unknown applications running in the background**.

- If you want to know the location of any specific ROS package/executable from in your system, you can use "rospack" find "package name" command. For example, if you would like to find 'lab2pkg_py' package, you can type in your console

  ```
  $ rospack find lab2pkg_py
  ```

- To move directly to the directory of a ROS package, use roscd. For example, go to lab2pkg_py package directory

  ```
  $ roscd lab2pkg_py
  ```

- Display Message data structure definitions with rosmsg

  ```
  $ rosmsg show <message_type>     #Display the fields in the msg
  ```

- rostopic, A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

  ```
  $ rostopic echo /topic_name      #Print messages to screen
  $ rostopic list          #List all the topics available
  $ rostopic pub <topic-name> <topic-type> [data...]
  #Publish data to topic
  ```

A.4 Create your own workspace

Since other groups will be working on your same computer, you should backup your code to a USB drive or cloud drive everytime you come to lab. This way if your code is tampered with (probably by accident) you will have a backup.

- Log on to the computer as 'ur3' with the password 'ur3'. If you log on as 'guest', you will not be able to use ROS.
- First create a folder in the home directory, mkdir catkin_(yourNETID). It is not required to have "catkin" in the folder name but it is recommended.

```
$ mkdir -p catkin_(yourNETID)/src
$ cd catkin_(yourNETID)/src
$ catkin_init_workspace
```

- Even though the workspace is empty (there are no packages in the 'src' folder, just a single CMakeLists.txt link) you can still "build" the workspace. Just for practice, build the workspace.

```
$ cd ~/catkin_(yourNETID)/
$ catkin_make
```

- **VERY IMPORTANT**: Remember to **ALWAYS** source when you open a new command prompt, so you can utilize the full convenience of Tab completion in ROS. Under workspace root directory:

```
$ cd catkin_(yourNETID)
$ source devel/setup.bash
```

A.5 Running a Node

- Once you have your catkin folder initialized, add the UR3 driver and lab starter files. The compressed file lab2andDanDriver.tar.gz, found at the class website contains the driver code you will need for all the ECE 470 labs along with the starter code for LAB 2. Future lab compressed files will only contain the new starter code for that lab. Copy lab2andDriverPy.tar.gz to your catkin directories "src" directory. Change directory to your "src" folder and uncompress by typing "tar -xvf lab2andDriver.tar.gz". You can also do this via the GUI by double clicking on the compressed file and dragging the folders into the new location.

  "cd .." back to your catkin_(yourNETID) folder and build the code with "catkin_make"

- After compilation is complete, we can start running our own nodes. For example our lab2node node. However, before running any nodes, we must have roscore running. This is taken care of by running a launch file.

```
$ roslaunch ur3_driver ur3_gazebo.launch
```

is used for the simulator.

```
$ roslaunch ur3_driver ur3_driver.launch
```

is used on the real robot.
This command runs both roscore and the UR3 driver that acts as a subscriber waiting for a command message that controls the UR3's motors.

- Open a new command prompt with "ctrl+shift+N", cd to your root workspace directory, and source it "source devel/setup.bash".
- We may also need to make lab2_exec.py executable.

  ```
  $ chmod +x lab2_exec.py
  ```

- Run your node with the command rosrun in the new command prompt. Example of running lab2node node in lab2pkg package:

  ```
  $ rosrun lab2pkg_py lab2_exec.py --simulator True
  ```

Note that the "--simulator True" tells it you are running on the simulator, while "--simulator False" tells the program you are running on real hardware. This flag is currently only applicable to Lab 2.

A.6 More Publisher and Subscriber Tutorial

Please refer to the webpage: `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c%2B%2B)`