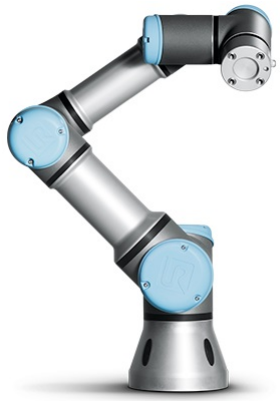
ECE 470
Introduction to Robotics
Lab Manual

ZJUI ver 1.0



Liangjing Yang
Songjie Xiao
Boyang Zhou
Shuren Li
Zhefan Lin
Zhenyu Zong

Zhejiang University/University of Illinois at
Urbana-Champaign Institute

UR3e Python - ROS Interface

Contents

LAB 4

Inverse Kinematics

4.1 Important

Read the entire lab before starting and especially the "Grading" section so you are aware of all due dates and requirements associated with the lab. Hopefully you are reading this well before your lab section meets as given the compressed schedule, it is very important that you arrive at lab well prepared.

4.2 Objectives

The purpose of this lab is to derive and implement a solution to the inverse kinematics problem for the UR3 robot. In this lab we will:

- Derive elbow-up inverse kinematic equations for the UR3
- Write a Python function that moves the UR3 to a point in space specified by the user.

4.3 Reference

Lecture slides provide examples of inverse kinematics solutions.

4.4 Tasks

4.4.1 Solution Derivation

Make sure to read through this entire lab before you start deriving your solution. There are some needed details not covered in this section.
Given a desired end-effector position in space $(x_{grip}, y_{grip}, z_{grip})$ and orientation $\{\theta_{yaw}, \theta_{pitch}(fixed), \theta_{roll}(fixed)\}$, write six mathematical expressions that yield

values for each of the joint angles. For the UR3 robot, there are many solutions to the inverse kinematics problem. We will implement only one of the *elbow-up* solutions.

- In the inverse kinematics problems you have examined in class (for 6 DOF arms with spherical wrists), usually the first step is to solve for the coordinates of the wrist center. The UR3 does not technically have a spherical wrist center but we will define the wrist center as z_{cen} which equals the same desired z value of the suction cup and x_{cen}, y_{cen} are the coordinates of θ_6's z axis. In addition, to make the derivation manageable, add that θ_5 will always be $-90°$ and θ_4 is set such that link 7 and link 9 are always parallel to the world x,y plane.

- Solve the inverse kinematics problem in the following order:

  1. x_{cen}, y_{cen}, z_{cen}, given yaw desired in the world frame and the desired x,y,z of the suction cup. The suction cup aluminum plate (link 9) has a length of 0.0535 meters from the center line of the suction cup to the center line of joint 6. Remember that this aluminum plate should always be parallel to the world's x,y plane. See Figure 4.2.
  2. θ_1, by drawing a top down picture of the UR3, Figure 4.1, and using x_{cen}, y_{cen}, z_{cen} that you just calculated.
  3. θ_6, which is a function of θ_1 and yaw desired. Remember that when θ_6 is equal to zero the suction cup aluminum plate is parallel to link 4 and link 6.
  4. x_{3end}, y_{3end}, z_{3end} is a point off of the UR3 but lies along the link 6 axis, Figure 4.1. For example if $\theta_1 = 0°$ then $y_{3end} = 0$. If $\theta_1 = 90°$ then $x_{3end} = 0$. First use the top down view of the UR3 to find x_{3end}, y_{3end}. One way is to choose an appropriate coordinate frame at x_{cen}, y_{cen} and find the translation matrix that rotates and translates that coordinate frame to the base frame. Then find the vector in the coordinate frame you chose at x_{cen}, y_{cen} that points from x_{cen}, y_{cen} to x_{3end}, y_{3end}. Simply multiply this vector by your translation matrix to find the world coordinates at x_{3end}, y_{3end}. For z_{3end} create a view of the UR3, Figure 4.2, that is a projection of the robot onto a plane perpendicular to the x,y world frame and rotated by θ_1 about the base frame. Call this the side view. Looking at this side view you will see that z_{3end} is z_{cen} offset by a constant.
  5. θ_2, θ_3 and θ_4, by using the same side view drawing just drawn above to find z_{3end}, Figure 4.2. Now that x_{3end}, y_{3end}, z_{3end} have been found use sine, cosine and the cosine rule to solve for partial angles that make up θ_2, θ_3 and θ_4. Hint: In this side view, a parallel to the base construction line through joint 2 and a parallel to the base construction line through joint 4 are helpful in finding the needed partial angles.

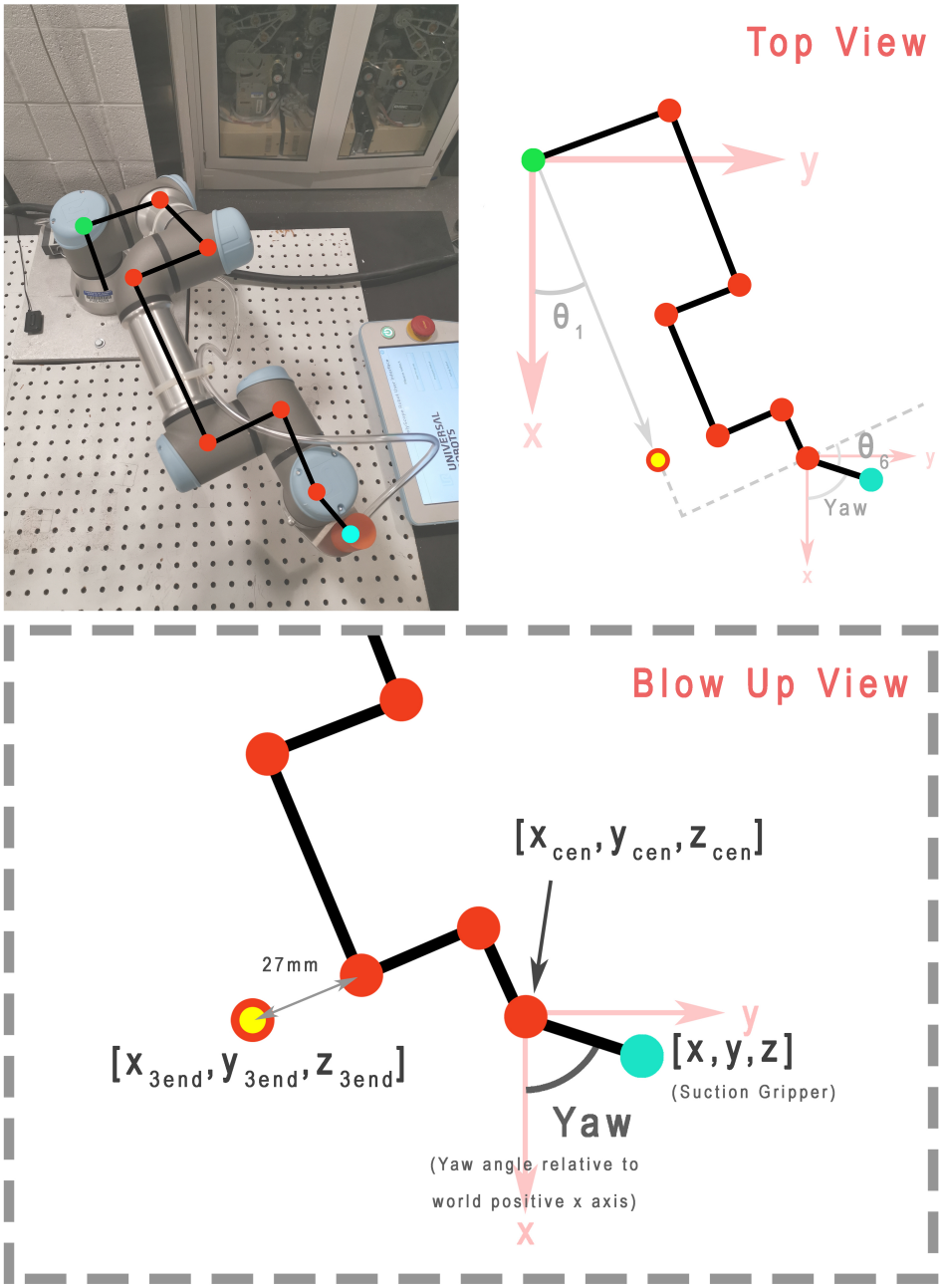


Figure 4.1: Top View Stick Pictorial of UR3. Note that the coordinate frames are in the same direction as the World Frame but not at the World frame's origin. One origin is along the center of joint 1 and the second is along the center of joint 6.

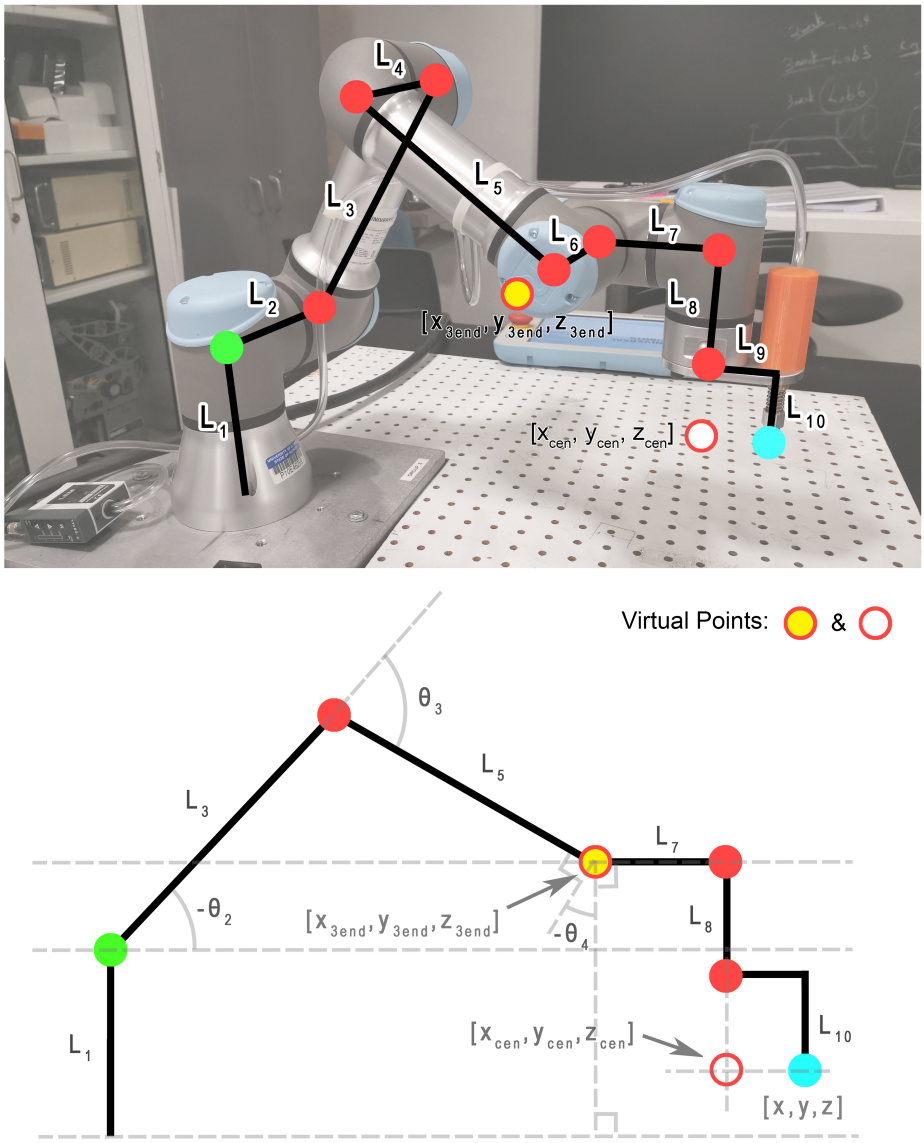


Figure 4.2: Side View Stick Pictorial of UR3.

4.4.2 Implementation

Implement the inverse kinematics solution by writing a Python function to receive world frame coordinates $(x_{Wgrip}, y_{Wgrip}, z_{Wgrip}, yaw_{Wgrip})$, compute the desired joint variables $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$, and command the UR3 to move to that pose using functions written in Lab4.

4.5 Procedure

- Download **lab4pkg_py.zip** from BlackBoard and extract it in your "**src**" directory. You will notice that there are three .py files. **lab4_exec.py**, **lab4_func.py** and **lab4_header.py**. The **lab4_func.py** file again will be compiled into a library so that future labs can easily call the inverse kinematic function. Like Lab 3, most of the needed code is given to you in **lab4_exec.py**. Your main job will be to add all the inverse kinematic equations to **lab4_func.py**. Please refer to the intermediate steps below to perform the inverse kinematic calculations. If you look at **lab4_header.py** it includes **lab4_header.py**. This allows you to call the functions you created in **lab4_func.py**.

- Once your code is finished, run it using "**rosrun lab4pkg_py lab4_exec.py [x] [y] [z] [yaw(degrees)]**" - e.g. "**rosrun lab4pkg_py lab4_exec.py 0.1 0.1 0.15 90**". Remember to run drivers in other command prompts at first.

- For in-person students, you should measure the x,y,z position of the end-effector using the provided ruler and square. For students using the simulation, it is not possible to measure this way, so we must use another method. A simple way is to use some of the ROS commands we learned before: "**rostopic echo /gripper/position -n 1**". These values are being calculated differently and so there will be small differences between this value and your calculations.

- You should verify that your code works by selecting a variety of poses that will test the full range of motion. Your TA will not be providing you test points.

- In your code (This is repeating the derivation steps above):

    1. Establish the world coordinate frame (frame w) centered at the corner of the UR3's base shown in Figure 4.3. The x_w and y_w plane should correspond to the surface of the table, with the x_w axis parallel to the sides of the table and the y_w axis parallel to the front and back edges of the table. Axis z_w should be normal to the table surface, with up being the positive z_w direction and the surface of the table corresponding to $z_w = 0$.
    We will solve the inverse kinematics problem in the base frame (frame

0), so we will immediately convert the coordinates entered by the user to base frame coordinates. Write three equations relating coordinates $(x_{Wgrip}, y_{Wgrip}, z_{Wgrip})$ in the world frame to coordinates $(x_{grip}, y_{grip}, z_{grip})$ in the base frame of the UR3.

$$x_{grip}(x_{Wgrip}, y_{Wgrip}, z_{Wgrip}) =$$
$$y_{grip}(x_{Wgrip}, y_{Wgrip}, z_{Wgrip}) =$$
$$z_{grip}(x_{Wgrip}, y_{Wgrip}, z_{Wgrip}) =$$

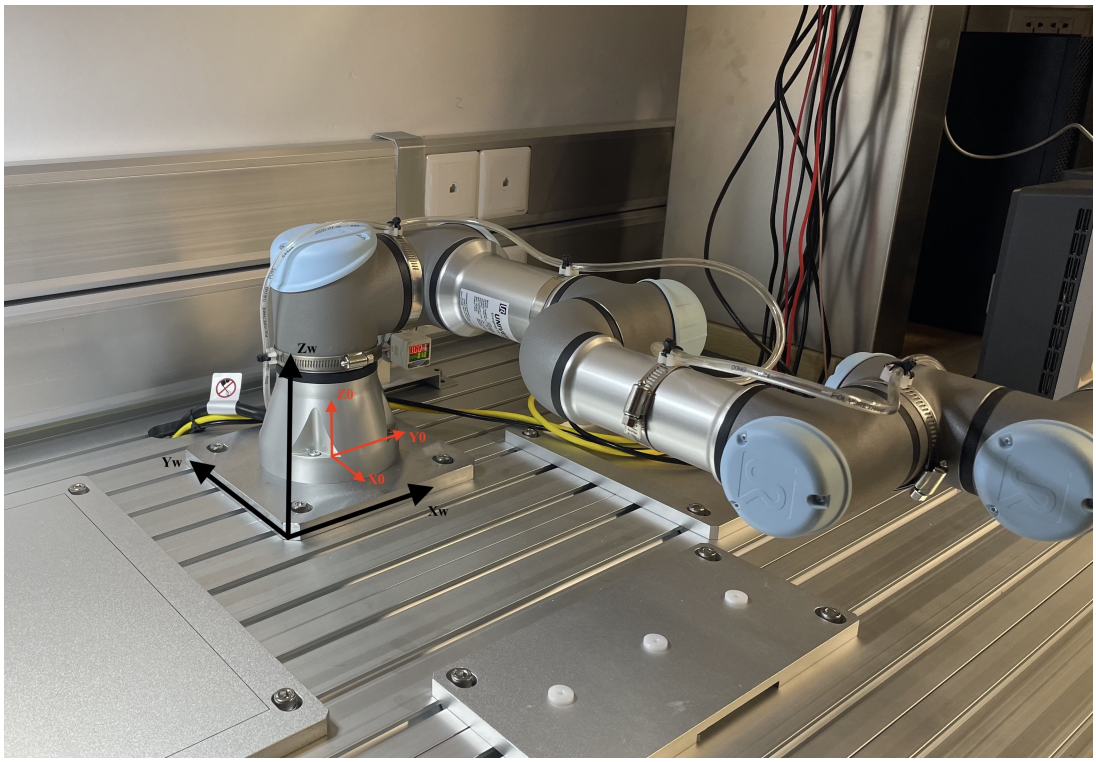


Figure 4.3: Correct location and orientation of the world frame.

2. Given the desired position of the gripper $(x_{grip}, y_{grip}, z_{grip})$ (in the base frame) and the yaw angle, find wrist's center point $(x_{cen}, y_{cen}, z_{cen})$.

$$x_{cen}(x_{grip}, y_{grip}, z_{grip}, yaw) =$$
$$y_{cen}(x_{grip}, y_{grip}, z_{grip}, yaw) =$$
$$z_{cen}(x_{grip}, y_{grip}, z_{grip}, yaw) =$$

3. Given the wrist's center point $(x_{cen}, y_{cen}, z_{cen})$, write an expression for the waist angle θ_1. Make sure to use the **atan2()** function instead

of **atan()** because **atan2()** takes care of the four quadrants the x,y coordinates could be in.

$$\theta_1(x_{cen}, y_{cen}, z_{cen}) = \tag{4.1}$$

4. Solve for the value of θ_6, given yaw and θ_1.

$$\theta_6(\theta_1, yaw) = \tag{4.2}$$

5. Find the projected end point $(x_{3end}, y_{3end}, z_{3end})$ using $(x_{cen}, y_{cen}, z_{cen})$ and θ_1.

$$x_{3end}(x_{cen}, y_{cen}, z_{cen}, \theta_1) =$$
$$y_{3end}(x_{cen}, y_{cen}, z_{cen}, \theta_1) =$$
$$z_{3end}(x_{cen}, y_{cen}, z_{cen}, \theta_1) =$$

6. Write expressions for θ_2, θ_3 and θ_4 in terms of the end point. You probably will want to define some intermediate variables to help you with these calculations.

$$\theta_2(x_{3end}, y_{3end}, z_{3end}) =$$
$$\theta_3(x_{3end}, y_{3end}, z_{3end}) =$$
$$\theta_4(x_{3end}, y_{3end}, z_{3end}) =$$

7. Now that your code solves for all the joint variables (remember that θ_5 is always $-90°$) send these six values to the Lab 3 function **lab_fk()**. You will need to copy you Lab 3 solution into these functions. Do this simply to check that your inverse kinematic calculations are correct. Double check that the x,y,z point that you asked the robot to go to is the same value displayed by the forward kinematic equations.

4.6 Report

You should submit a lab report using the guidelines given in the ECE 470: How to Write a Lab Report document. Please be aware of the following:

- **Lab reports are due before Lab5!**
- Lab reports will be submitted online at BlackBoard.

Your lab report should include the following:

- A clearly written derivation of the inverse kinematics solution for each joint variable $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$. You **must** include figures in your derivation. Diagrams should be your own creation and clear and easily read. Do not use hand drawn figures or annotations.
- For each test point include:
  - The given $\{\left(x_{w_{grip}}, y_{w_{grip}}, z_{w_{grip}}\right), \theta_{yaw}\}$
  - The measured position
  - The scalar error
- Include a brief discussion of sources of error.

As appendices to your report, include the following:

- Your **lab4_func.py** code and **lab4_exec.py** if it was edited.

4.7 Demo

Your TA will require you to run your program twice, each time with a different set of desired position and orientation. Your program should reach the desired position and orientation with almost no error. You will be required to be able to demo on the simulator, even if you choose to demo on the real robot.

4.8 Grading

- 80% successful demonstration.
- 20% individual report.

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.

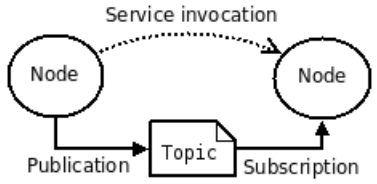


Figure A.1: source: `http://wiki.ros.org/ROS/Concepts`

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)`