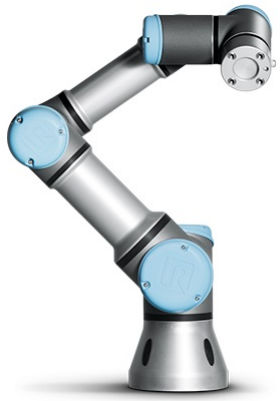
ECE 470
Introduction to Robotics
Lab Manual

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

3 Forward Kinematics **1**
- 3.1 Important . . . 1
- 3.2 Objectives . . . 1
- 3.3 References . . . 1
- 3.4 Tasks . . . 2
  - 3.4.1 Theoretical Solution . . . 2
  - 3.4.2 Physical Implementation . . . 2
  - 3.4.3 Comparison . . . 2
- 3.5 Procedure . . . 2
  - 3.5.1 Theoretical Solution . . . 2
  - 3.5.2 Implementation on UR3 . . . 5
  - 3.5.3 Comparison . . . 6
- 3.6 Report . . . 7
- 3.7 Demonstration . . . 8
  - 3.7.1 Demo Process . . . 8
- 3.8 Grading . . . 8
- 3.9 Tentative Due Dates - Fall 2020 . . . 8
  - 3.9.1 Group A . . . 8
  - 3.9.2 Group B . . . 9
- 3.10 Preparation for Lab 4 . . . 9

A ROS Programming with Python **13**
- A.1 Overview . . . 13
- A.2 ROS Concepts . . . 13
- A.3 Before we start.. . . . 14
- A.4 Create your own workspace . . . 16
- A.5 Running a Node . . . 16
- A.6 More Publisher and Subscriber Tutorial . . . 17

LAB 3

Forward Kinematics

3.1 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.

3.2 Objectives

The purpose of this lab is to implement the forward kinematics on the UR3 robot to estimate the end-effector pose given a set of joint angles. In this lab you will:

- Solve the forward kinematic equations for the UR3.
- Write a Python function that moves the UR3 to a configuration specified by the user.
- Compare your forward kinematic estimation with the actual robot movement.

3.3 References

- Lab 3 supplementary material on Blackboard provides details of how to construct a forward kinematic for an open-chain robot using PoE.

3.4 Tasks

3.4.1 Theoretical Solution

Find the forward kinematic equations for the UR3 robot using the forward kinematics method. Solve for T_{06} and record the 6 matrices defined by $e^{[S_1]\theta_1}$ through $e^{[S_6]\theta_6}$.

3.4.2 Physical Implementation

The user will provide six joint angles $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$, all given in degrees. The angle ranges are as follows:

$$\begin{aligned} -90^\circ < \quad & \theta_1 \quad < 90^\circ \\ -180^\circ < \quad & \theta_2 \quad < 0^\circ \\ -5^\circ < \quad & \theta_3 \quad < 140^\circ \\ -95^\circ < \quad & \theta_4 \quad < -5^\circ \\ -115^\circ < \quad & \theta_5 \quad < 70^\circ \\ -170^\circ < \quad & \theta_6 \quad < 170^\circ \end{aligned}$$

Note that these ranges are approximate and vary slightly from machine to machine. If you receive a protective stop, note the affected joint on the teach pendant and make appropriate corrections. It is also worth noting that not all configurations are achievable or desirable. Some will strike the table or wall or be difficult to measure accurately. These limits are primarily for the real robot and will not affect the simulation, but it is advised that you make use of them to achieve realistic configurations.
Once the angles are selected, using the ROS Python program move the joints to these desired angles. Your program should additionally calculate and display the translation matrix T_{06} that rotates and translates the end effector's coordinate frame to the base frame.

3.4.3 Comparison

For any provided set of joint angles $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$, compare the measured position or the position output from the ROS topic with the value calculated by your Python forward kinematics function.

3.5 Procedure

3.5.1 Theoretical Solution

1. In exponential forward kinematics (especially for an open chain kinematics), there are three components that are essential for the calculation.

- A fixed base frame f_0
- Robot's "zero" configuration (or some might call it initial position)
- A well-defined end-effector frame f_6

Practically, you can define the base frame, the end-effector frame, and zero configuration arbitrarily. As long as you remain consistent with your definitions, the end results will be identical for the same robot. However, doing so will pose difficulties when your TA tries to verify your answer or debug your process. Therefore we encourage you to use the same frame placement and zero configuration for both convenience and safety purposes. See Figure 3.1. For convenience when measuring, the base frame, f_0, is placed at the left corner of the aluminum base plate which the UR3 is mounted on. It is even with the peg-board surface.

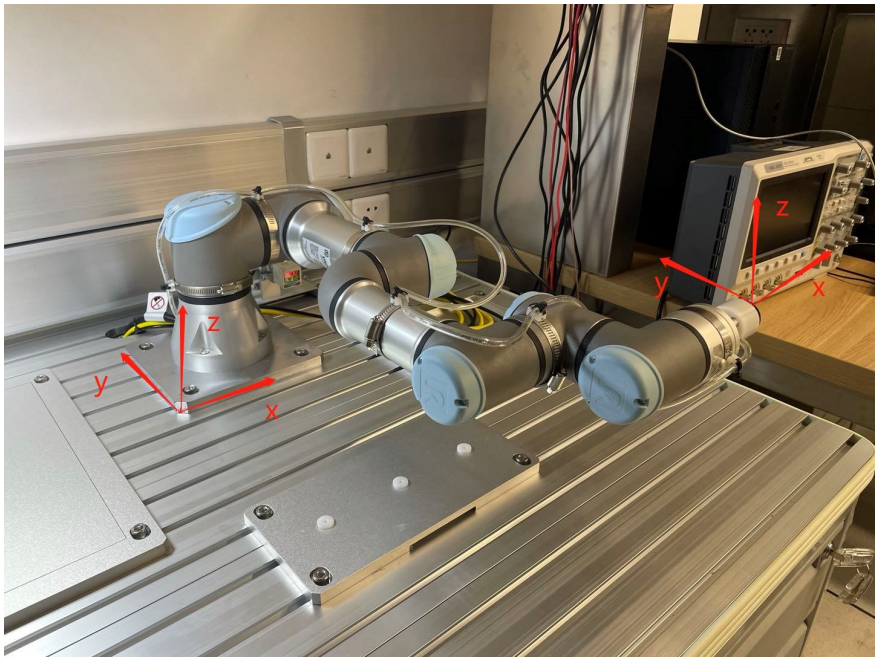


Figure 3.1: "zero" configuration and frame placement for UR3.

Moreover, please note that in this lab, all the UR3 robots have a predefined zero configuration. The pose shown in Figure 3.1 is **NOT the TRUE zero configuration** according to the teach pendant. In this pose (Figure 3.1), you would read the following joint angles from **the teach pendant**:

Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
90°	0.0°	0.0°	-90.0°	0.0°	0.0°

In the **teach pendant's** zero configuration, (due to how the robot is originally mounted) the robot arm would face the far edge of the table and the wrist joint #2 would point into the table, which is an awkward pose to make measurements and do experiments. Hence, we added some angle offsets in the function script such that:

```
return_value[0] = theta1 + (0.5*PI)
return_value[1] = theta2
return_value[2] = theta3
return_value[3] = theta4 - (0.5*PI)
return_value[4] = theta5
return_value[5] = theta6
```

(**Attention:** Remember in the previous lab, when we enter the angle value on control panal into python code, the value of base and elbow should be switched.)
Any joint angle measurements from the robot that are passed to your ROS program will be corrected to the "zero" configuration shown in Figure 3.1. In other words, now if you set all the joint of UR3 to zero (zero configuration) in **ROS**:

Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
0.0°	0.0°	0.0°	0.0°	0.0°	0.0°

You would see the UR3 poses as Figure 3.1 showed.

2. The sole purpose of the forward kinematics algorithm is to estimate the end-effector's pose (orientation and position) with respect to the base frame given a set of joint angles. This information can be packed and calculated in a "homogeneous transformation matrix" as taught by Lynch & Park in *Modern Robotics*:

$$T(\theta) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} ..... e^{[S_{n-1}]\theta_{n-1}} e^{[S_n]\theta_n} M$$

While our objective is to solve for the homogeneous transformation matrix $T(\theta)$ of the "motion" caused by certain joint angle configurations, there are seven matrices we need to specify:

- The screw axes S_1 to S_6 (we have six here because UR3 has six joints) expressed in the base frame, corresponding to the joint motions when the robot is at its zero configuration (see Figure 3.1)
- The end-effector pose M (a 4x4 homogeneous transformation matrix) with respect to the base frame, corresponding to the zero configuration

Figure 3.2: The "right hand rule"

A screw axis S can be represented by $S = (\omega, v)$, where the ω is the joint axis defined by the **right hand rule** (Figure 3.2). v can be computed by $v = -\omega \times q$, where q is the displacement (position) of the joint center with respect to the base frame.

3. Using the symbolic toolbox of Matlab or Python, find the closed form solution of T_{06}. You should note how long and complex the solution is due to the contribution of each joint's motion. Additionally, calculate the 6 matrices defined by $e^{[S_1]\theta_1}$ through $e^{[S_6]\theta_6}$.

4. To aid in your calculations, several figures have been included.

   - Figure 3.3 shows the screw axes and direction of rotation (use the right rule) for each joint.
   - Figure 3.4 shows the joint centers to aid in locating your q values.
   - Figure 3.5 shows the dimensions of all of the links of the UR3.
   - Figure 3.6 shows the dimensions of the end effector including the aluminum plate and suction gripper.
   - Figure 3.7 shows the offset between the world frame and the base of the UR3.

3.5.2 Implementation on UR3

1. Download and extract **lab3Py.tar.gz** into the "**src**" directory of your catkin directory. Don't forget "**source devel/setup.bash**". Then from

your base catkin directory run "**catkin_make**" and if you receive no errors you copied your lab3 starter code correctly. Also so that ROS registers this new package "**lab3pkg_py**", run "**rospack list**" and you should see **lab3pkg_py** as one of the many packages.

2. You will notice that in **lab3pkg_py/scripts** there are now three *.py files. **lab3_exec.py** is the main() code, **lab3_func.py** defines the important forward kinematic functions. **lab3_header.py** defines the header files. We divide them up in this fashion for clarity. In this lab you will be mainly editing **lab3_func.py**. You can of course change **lab3_exec.py** but most of its functionality has already been given to you. Study the code and comments in **lab3_exec.py** and **lab3_func.py** to see what the starter code is doing and what parts you are going to need to change. Your job is to add the code in the function **Get_MS()** that correctly populates the six screw axes S_1 to S_6 as well as the transformation matrix M. The Python code uses the "numpy" module to create and multiply matrixes; meanwhile, the matrix exponential "expm()" is achieved by including the "Scipy" module. Then, with the S and M values you populated, write the code in function **lab_fk()** that generates and prints the homogeneous transformation matrix $T(\theta)$.

3. Once your code is finished, run it using "**rosrun lab3pkg_py lab3_exec.py [theta1] [theta2] [theta3] [theta4] [theta5] [theta6]**" with all angles in degrees - e.g. **rosrun lab3pkg_py lab3_exec.py 0 90 45 0 -90 105**. Remember that in another command prompt you should have first run roscore and drivers using "**roslaunch ur3_driver ur3_gazebo.launch**" or "**roslaunch ur3_driver ur3_driver.launch**" depending if you are using the simulator or real robot.

4. 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.

5. 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 angles, so you should make use of the joint limits provided in Section 3.4.2.

3.5.3 Comparison

1. Your TA will select two sets of joint angles $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$ when you demonstrate your working ROS program.

2. Run your ROS node and move the UR3 to each of these configurations. With a ruler and square (or the appropriate ROS command for the simulation), measure the x,y,z position vector of the center of the gripper for each set of angles. Call these vectors r_1 and r_2 for the first and second sets of joint angles, respectively.

3. Compare these measurements to the translation vector calculated by your ROS node. Call these calculated vectors d_1 and d_2.

4. For each set of joint angles, Calculate the error between the measured and calculated kinematic solutions. We will consider the error to be the magnitude of the distance between the measured center of the gripper and the location predicted by the forward kinematic equation:

$$error_1 = \|r_1 - d_1\| = \sqrt{(r_{1x} - d_{1x})^2 + (r_{1y} - d_{1y})^2 + (r_{1z} - d_{1z})^2}.$$

   A similar expression holds for $error_2$.

3.6 Report

Each student will 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 will be submitted online at Blackboard.
- The report will be due **2** weeks after your lab session for Lab 3. Exact times and dates can be seen on Blackboard.

Your lab report should include the following:

- Explain how you determined $\mathcal{S}_1$ thru $\mathcal{S}_6$ and $\mathbf{M}$ - include figures as needed.
- Include a figure that shows all frames and axes used.
- Include a table of all ω and q used and the v derived.
- Figures should be your own creation and not copies of the lab manual.
- Use Matlab or Python's symbolic toolbox to generate the final homogeneous transform T_{06} as a function of $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$. As this is too cumbersome to include in your report, include $e^{[S_1]\theta_1}$ through $e^{[S_6]\theta_6}$ in your report (do not use screenshots).
- Substitute in your given joint angles into your symbolic solution to verify it.
- For each test point, include:
  - The given $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$

  - The calculated d and measured r vectors
  - The scalar error
  - The output of your symbolic solution
- Include a brief discussion of sources of error.

Unless your TA gives other guidance, include the following as appendices:

- Your **lab3_func.py** code (and **lab3_exec.py** if it was edited).
- Your Matlab or Python code used to calculate the symbolic T_{06}.

3.7 Demonstration

Demonstrations of your working code will either be done in-person or over Zoom. They will be done live (i.e. no recordings) with your lab section TA. Demos are due **2** weeks after your lab session.

The default method of demonstrating your work will be via the simulation. While we want to encourage in-person students to run their code on the real robot, due to time limitations, this may be difficult to do with your TA by the due date. You can always run it for your own satisfaction at another time.

3.7.1 Demo Process

Your TA will require you to run your program twice, each time with a different set of joint variables.

3.8 Grading

- 80 points, successful demonstration.
- 20 points, individual report.

3.9 Tentative Due Dates - Fall 2020

The exact date and time will depend on your lab section and can be found on GradeScope.

3.9.1 Group A

- Demonstration - Week of October 19
- Report - Week of October 26

3.9.2 Group B

- Demonstration - Week of October 26
- Report - Week of November 2

3.10 Preparation for Lab 4

Lab 4 will cover inverse kinematics. In class you will mostly focus on numerical inverse kinematics, but we will be using analytical inverse kinematics - i.e. we will use geometry and constraints to find a closed form solution to the problem. Suggested Reading:

- Chapter 6 of *Modern Robotics* provides multiple examples of inverse kinematics solutions. Especially Section 6.1.
- Chapter 3 of *Robot Modeling and Control*, by M. Spong, S. Hutchinson and M Vidyasagar (Wiley and Sons, 2005). Especially Section 3.3.3. (A version of this text is available online.)

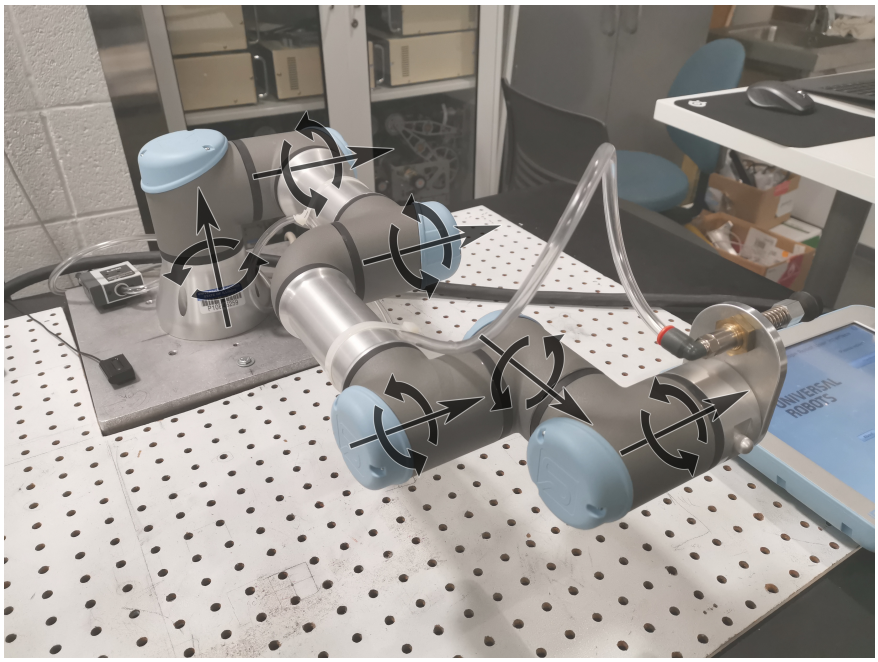


Figure 3.3: Screw axes on UR3.

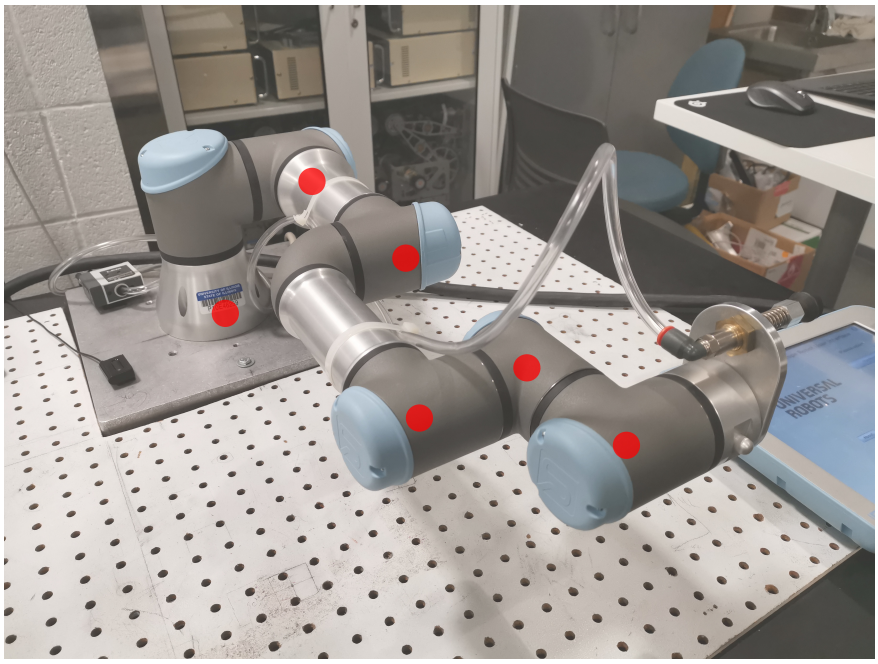


Figure 3.4: Joint center locations on UR3.

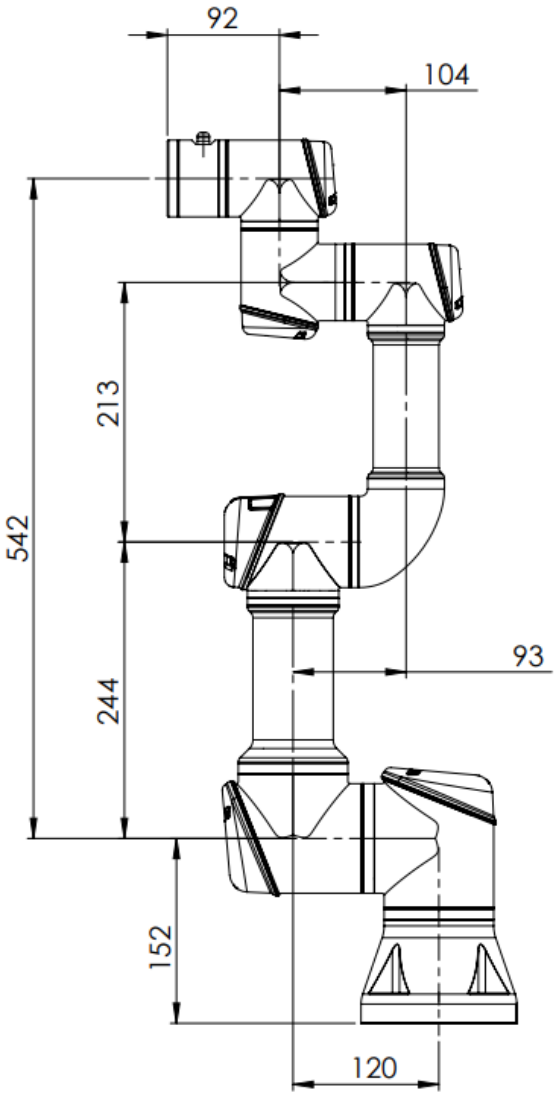


Figure 3.5: Approximate Dimensions of the UR3 in mm. *EACH UR3 IS SLIGHTLY DIFFERENT DUE TO ITS MANUFACTURE TOLERANCE.

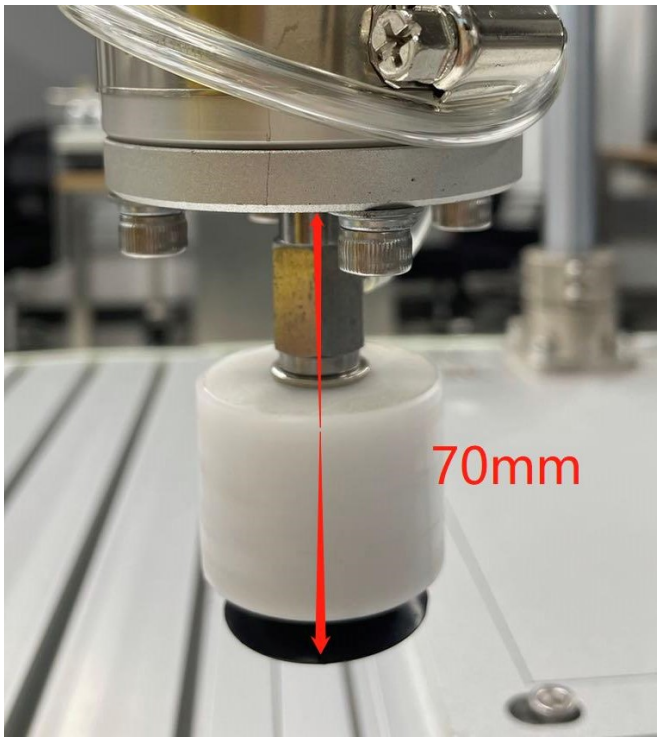


Figure 3.6: Approximate Dimensions of the attached end effector plate and suction gripper in mm.

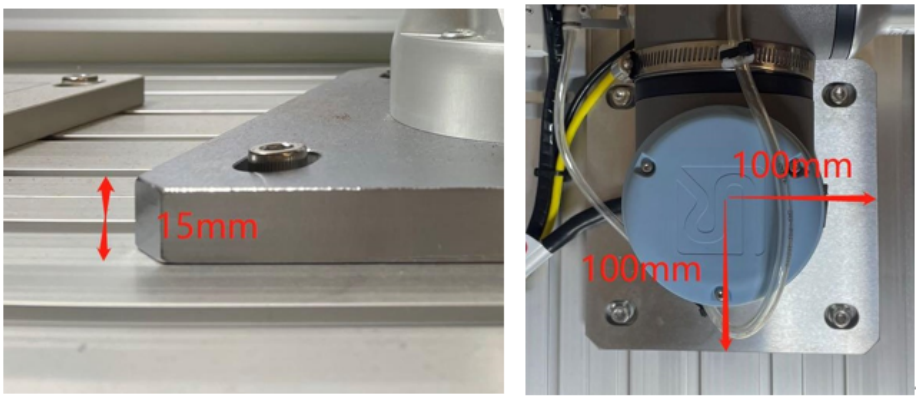


Figure 3.7: Approximate Dimensions of the relationship between the world frame and the base of the UR3 in mm.

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.

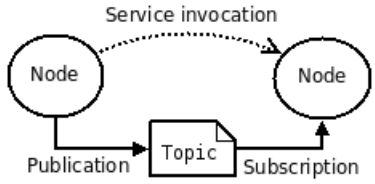


Figure A.1: source: `http://wiki.ros.org/ROS/Concepts`

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)`