LAB 1

Introduction to the UR3

1.1 Lab Introduction

1.1.1 Important
Read the entire lab before starting and especially the “Grading” section so you know what needs to be accomplished before finishing the lab.

1.1.2 Objectives
The purpose of this lab is to familiarize you with the UR3 robot arm and use psuedo code to finish a simple block detection and placement job. In this lab, you will:

- Learn the UR3 robots basic setup and use the teach pendent language to create a simple psuedo program for the UR3.
- Learn how to use the teach pendant language to turn on and off the suction cup gripper and use the gripper in a program
- Write a psuedo code using teach pendent language that places one block on top of another.

1.1.3 References
- UR3 Owner’s Manual:
  https://www.universal-robots.com/download/?option=52870#section52851
- UR3 Software Manual:
  https://www.universal-robots.com/download/?option=53077#section53064
1.2. LAB ACTIVITY

- Universal Robots Academy
  
  https://www.universal-robots.com/academy/

1.1.4 Pre-Lab

Before you come to lab it is very important that you go through the training videos found at Universal Robots website https://www.universal-robots.com/academy/. 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 Lab Activity

1.2.1 Task

Using the teach pendant psuedo code, each team will “program” the UR3 to pick and place blocks. The program may do whatever you want, but all programs must check three predefined locations for two blocks and stack one block on top of another at a fourth predefined position. You will use the gripper’s suction feedback to determine if a block is located at one of the three starting block locations. The blocks must be aligned with each other in the stack of two.

1.2.2 Procedure

1. The Pre-Lab asked you to go through the basic UR3 training at Universal Robots website. This training should have shown you how to make simple programs to move the UR3.

2. To turn on the suction for the suction cup gripper, Digital output 0 needs to be set high. Set low to turn off the suction. Also Digital input 0 indicates if the suction cup is gripping something. It will return 1 if it is gripping an object and 0 if not. Modify your above program (or make a new one) to add activating on and off the suction cup gripper.

3. Create a program that defines four spots on the robot’s table. Three of these spots are where it is possible a block will be initially located and with a certain orientation. There will only be two blocks. The user will place the blocks in two of the positions. The goal for the robot is to collect the two blocks and stack them on top of each other in the fourth define place on the robot’s table. So you will need to use the suction cup gripper’s feedback that indicates whether an object is being gripped or not. Then with some “If” instructions complete this task such that the user can put the two blocks in any of the three starting positions. When you are finished, Your psuedo code should convince your TA that
your program works when two blocks are placed and aligned in the three
different configurations and also does not have a problem if only one block
or even no blocks are placed at their starting positions. Tips for creating
this program when we use the teach pendent to finish the job:

- To turn on the suction cup, use the Set command and select Digital
  Output 0 and turn it on or true. Set it to off or false to turn off the
  suction.
- Digital Input 0 indicates if something has been gripped by the
  suction cup. Go to the I/O tab and turn on and off Digital Output
  0 and check which state of Digital Input 0 indicates gripped and
  ungripped.
- In the Structure tab under Advanced besides “If ... else”, you may
  also want to use the Assignment to create a global worker variable
  that, for example, stores the number of blocks collected. In addition
  the SubProg item creates a subroutine that you may call when
  performing the same steps. The subroutine’s scope allows it to see
  the variables you create with the Assignment item.
- You may want to name your Waypoints. This makes your program
  easier to read. In addition if the robot needs to go to the same point
  multiple times in your program you can command it to go to the
  same waypoint name.
- Under the Structure tab you can use the Copy and Paste buttons
to copy a line of code and past it in a different subsection of your
  code. This cuts down on extra typing. Also note the Move up and
down buttons along with the Cut and Delete buttons. Suppress is
  like commenting out a line of code.
- When you add an “If” statement and then click on the Command
tab, tap in the long white box to pull up the keyboard for entering
  the if condition.

4. You should formulate a psuedo code that could accomplish the task
described above. You could also add some comments to make your psuedo
code clear and simple.

1.3 Lab Deliverable and Grading for Online Students

1.3.1 Report

You will submit a report that uses pseudo-code to describe how you would solve
the task described in Section 1.2.

- Please organize your code using Python style indentation. If you are
  unfamiliar with Python, please discuss this with your TA.
1.4. PREPARATION FOR LAB 2

- Try to use clear terminology similar to what you learned in the Academy e.g. Move, If... Else..., Set, etc.

- Your TA will inform you how and when they wish you to deliver the completed report (Gradescope/Email).

- There is no specified format for the report as long as it is typed and well organized. But you should explain your pseudo code format before using it. Please submit as a pdf.

We understand that this task is a bit abstract and may not be easy to visualize without the robot in front of you. Try your best and think about what you learned in the Academy.

1.3.2 Demo

None required.

1.3.3 Grading

- 100 points for satisfactorily completing the report.

1.4 Preparation for Lab 2

In Lab 2, we will be moving on to use ROS to control the UR3. We will use it to solve a modified version of the famous Towers of Hanoi problem. Some suggested reading:

- Consult Appendix A of this lab manual for details of ROS and Python functions used to control the UR3.

  - Chapter 2: 2.4 Packages, 2.5 The Master, 2.6 Nodes, 2.7.2 Messages and message types.
  - Chapter 3 Writing ROS programs.


- 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. You are not required to implement a recursive solution.

---

[1] [http://www.cut-the-knot.org/recurrence/hanoi.shtml](http://www.cut-the-knot.org/recurrence/hanoi.shtml) (an active site, as of this writing.)
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/](http://wiki.ros.org/)
- How to install on your own Ubuntu: [http://wiki.ros.org/ROS/Installation](http://wiki.ros.org/ROS/Installation)
- For detailed tutorials: [http://wiki.ros.org/ROS/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
A.3. BEFORE WE START..

the first four.

- **Nodes** “programs” or ”processes” in ROS that perform computation. For example, one node controls a laser range-finder, one node controls the 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:
  
  $\texttt{roscore}$

  roscore can also be started automatically when using roslaunch in terminal, for example:
  
  $\texttt{roslaunch <package name> <launch file name>}.launch$

  # the launch file for all our labs:

  $\texttt{roslaunch ur3\_driver ur3\_driver\_launch}$

- **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.

![Diagram](http://wiki.ros.org/ROS/Concepts)

Figure A.1: source: [http://wiki.ros.org/ROS/Concepts](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.

  $\texttt{ls}$
A.3. BEFORE WE START..

- 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.

- First create a folder in the home directory, mkdir catkin\(\text{(yourNETID)}\). It is not required to have “catkin” in the folder name but it is recommended.

  $ \text{mkdir -p catkin}\_\text{(yourNETID)}/\text{src}$
  $ \text{cd catkin}\_\text{(yourNETID)}/\text{src}$
  $ \text{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.

  $ \text{cd } /\text{catkin}\_\text{(yourNETID)}$
  $ \text{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:

  $ \text{cd catkin}\_\text{(yourNETID)}$
  $ \text{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 -zxvf lab2andDriver.tar.gz”.

  “cd ..” back to your catkin\(\text{(yourNETID)}\) 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.

  $ \text{roslaunch ur3\_driver ur3\_driver\_launch}$

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 also need to make lab2_exec.py executable.

  $ chmod +x lab2_exec.py

• Run your node with the command rosrut in the new command prompt. Example of running lab2dannode node in lab2danpkg package:

  $ rosrut lab2pkg_py lab2_exec.py

---

A.6 More Publisher and Subscriber Tutorial

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