The Chinese University of Hong Kong
Department of Electronic Engineering
2023-24 Term 2

ELEG4701 Intelligent Interactive Robot Practice
Laboratory 8: Lidar-based Navigation

Objective

In this lab session, you will learn:

1. basic knowledge of lidar and point cloud
2. How to create a 3D env by yourself
3. Basic feature extraction method
4. Point cloud reconstruction method
5. A path planning algorithm

Name: ______________________

SID: ______________________

Date: ______________________

Please return this lab sheet to TAs before you leave.

Cheat Sheet

Basic command-line tools:

- Use Ctrl + Shift + T to open a new page on a terminal.
- Use Tab to automatically complete the command, filename, etc.
- Go to a directory `cd <directory-path>`
- List the items in the current directory: `ls`
- Create a new directory: `mkdir <directory-name>`
- Create a new file: `touch <file-name>`
- Go to a ROS package directory: `roscd <package-name>`
- Make the catkin workspace after changing ROS packages: `catkin_make`
- Source the environment after `catkin_make`: `source devel/setup.bash`
- Make a Python node executable: `chmod +x node.py`
- Run a ROS node: `rosrun <package-name> <node-name> [args]`
- List the available ROS nodes / topics / services: `rosnode list` `rostopic list` `rosservice list`
- The `roslaunch` tool can help you easily bring up a set of ROS nodes together. A `roslaunch` will automatically start `roscore` if it is not already running.
- roslaunch command-line usage: `roslaunch <package-name> <launch-filename>`.
  e.g., “`roslaunch rospy_tutorials talker_listener.launch`”
- roslaunch .launch file format example:

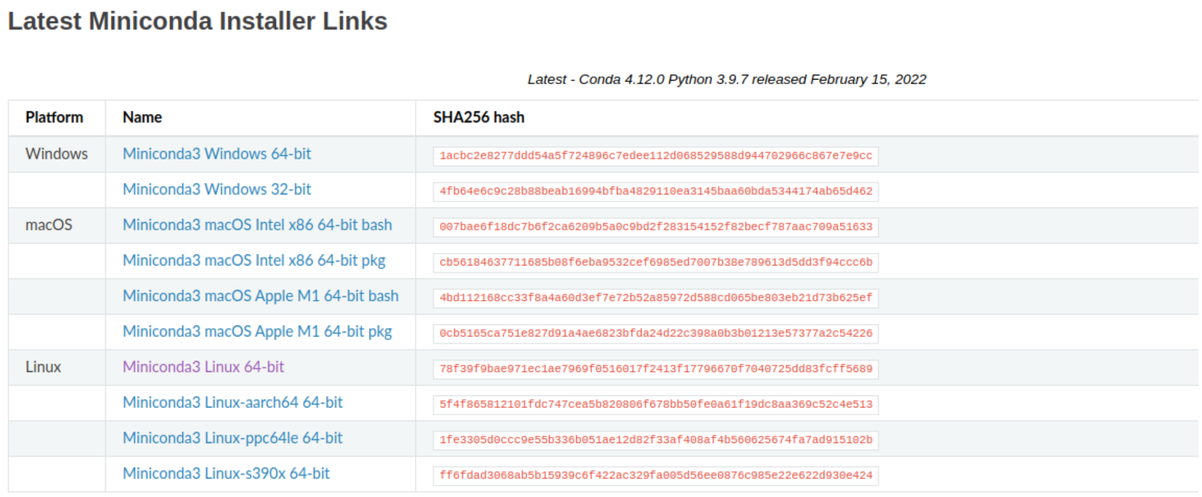
  ```
  <launch>
    <node name="talker1" pkg=“rospy_tutorials" type="talker.py" />
    <node name="listener1" pkg="rospy_tutorials" type="listener.py"/>
  </launch>
  ```

  This example will launch the “talker1” node using the talker.py executable and the “listener1” node using the listener.py executable from the rospy_tutorials package.
- An easy-to-use and general-purpose text editor: `gedit <file-name>`
- Some tips:
  - Use `rosnode`, `rostopic`, `rosservice` commands to debug.
  - Google your errors and use ROS wiki to search for the usage of ROS packages.

Task 0: Prepared for Programming

Please follow the guide below to prepare the tools needed in this lab:

- Install miniconda (miniconda is a very useful environment manager software for Python)
  https://docs.conda.io/en/latest/miniconda.html#install_miniconda



Latest Miniconda Installer Links

Latest - Conda 4.12.0 Python 3.9.7 released February 15, 2022

Platform	Name	SHA256 hash
Windows	Miniconda3 Windows 64-bit	1acbc2e8277ddd54a5f724896c7edee112d068529588d944702966c867e7e9cc
	Miniconda3 Windows 32-bit	4fb64e6c9c28b88beab16994bfba4829110ea3145baa60bda5344174ab65d462
macOS	Miniconda3 macOS Intel x86 64-bit bash	007bae6f18dc7b6f2ca6209b5a0c9bd2f283154152f82becf787aac709a51633
	Miniconda3 macOS Intel x86 64-bit pkg	cb56184637711685b08f6eba9532cef6985ed7007b38e789613d5dd3f94ccc6b
	Miniconda3 macOS Apple M1 64-bit bash	4bd112168cc33f8a4a60d3ef7e72b52a85972d588cd065be803eb21d73b625ef
	Miniconda3 macOS Apple M1 64-bit pkg	0cb5165ca751e827d91a4ae6823bfda24d22c398a0b3b01213e57377a2c54226
Linux	Miniconda3 Linux 64-bit	78f39f9bae971ec1ae7969f0516017f2413f17796670f7040725dd83fcff5689
	Miniconda3 Linux-aarch64 64-bit	5f4f865812101fdc747cea5b820806f678bb50fe0a61f19dc8aa369c52c4e513
	Miniconda3 Linux-ppc64le 64-bit	1fe3305d0ccc9e55b336b051ae12d82f33af408af4b560625674fa7ad915102b
	Miniconda3 Linux-s390x 64-bit	ff6fdad3068ab5b15939c6f422ac329fa005d56ee0876c985e22e622d930e424

- Open your terminal and run the commands below:

  `$cd ~/Downloads`

  `$chmod +x Miniconda3-latest-Linux-x86_64.sh`

  `$sh Miniconda3-latest-Linux-x86_64.sh` #press Enter and input yes to install miniconda

  Close the terminal and reopen a new one.

  `$conda config --set auto_activate_base false`

  `$conda create -n lab8 python=3.8` #create a virtual environment named lab8

- Download lab8_project.rar from Blackboard, extract it in your home dir

  `$cd ~/lab8_project/` #activate lab8 virtual environment

  `pip install -r package.txt` #install all the packages in "package.txt"

  If pygame installation failed, try `pip install pygame --pre`

  If missing testresources, try `$sudo apt install python3-testresources`

1 Task 1: Simulate a Lidar (10%)

Read `Lidar_LabCode_Tutorial.pdf` in `lab8_project`.

1. Read the code in “`learn_lidar.py`” in `lab8_project/lidar3d/` and finish TODO:
   Adding 2 parameters to makeSphere functions, including position and radius.
   Hint: for sparse and dense functions, set radius to 20; for cut off function set radius to 1.
2. Run “`learn_lidar.py`”

After you finish this task, please show it to the TA.

Checked by TA: ________________________

Finished Steps: ____________/2____________

2 Task 2: Make a Stage (30%)

Read `Lidar_LabCode_Tutorial.pdf` in `lab8_project`.

1. Read the code in “`Stage.py`” in `lab8_project/lidar3d/` and finish TODO in `make_your_stage3d`. Create **multiple cubes** and **at least one sphere** as your own stage. You need to use `utils.makeCube`, `utils.makeSphere`, and `utils.transModel`. Do not forget to change the main function to view the default stage or your own stage.
2. Run “`Stage.py`”
3. Read the code in “`LidarCore.py`” in `lab8_project/lidar3d/` and finish TODO in `try_lidar`. Locate lidar in your stage and do consider the range of lidar. Hint: Using `self.setLidarPosition`.
4. Run “`LidarCore.py`”
5. Read the code in “`AnimePlayer.py`” in `lab8_project/lidar3d/` and finish TODO in `test_anime_your_model`. Set the path of the model and do some modification.
6. Run “`AnimePlayer.py`” 3 times, each time with a different main function. (1.default stage 2.your own stage 3.reconstruction)

After you finish this task, please show it to the TA.

Checked by TA: ________________________

Finished Steps: ____________/6____________

3 Task 3: Semantic Segmentation (30%)

Read `Lidar_LabCode_Tutorial.pdf` in `lab8_project`.

1. Read the code in "`PointsSegmentation.py`" in `lab8_project/lidar3d/` and finish TODO in function `points2Image`. Create codes for dbscan to construct correct obj_map.
2. Run "`PointsSegmentation.py`" and compare your rule with the machine learning algorithms. Save your rules' obj_map and dbscan's obj_map with different names(`obj_map.obj` for your rule & `obj_map_dbscan` for dbscan). You can do it by rename or a single line of code. Show both of them to TA.
3. After step 2, Check if `obj_map.bmp` shows in you `lab8_project/lidar3d/`.
4. Finish TODO in"`rrt_mystage.py`". Set the target position and draw the trajectory with at least 2 turns. Hint: you may need to try multiple times, since each time RRT path might be different. Otherwise, you may need to change the start position or even your own stage to meet the requirements.

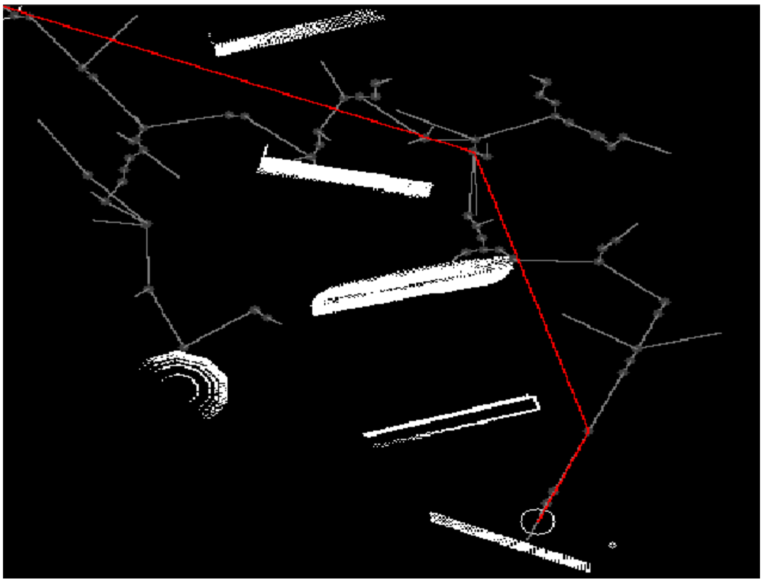


Figure 1: You should have got something like this for your Task 3.

After you finish this task, please show it to the TA.

Checked by TA: ______________________

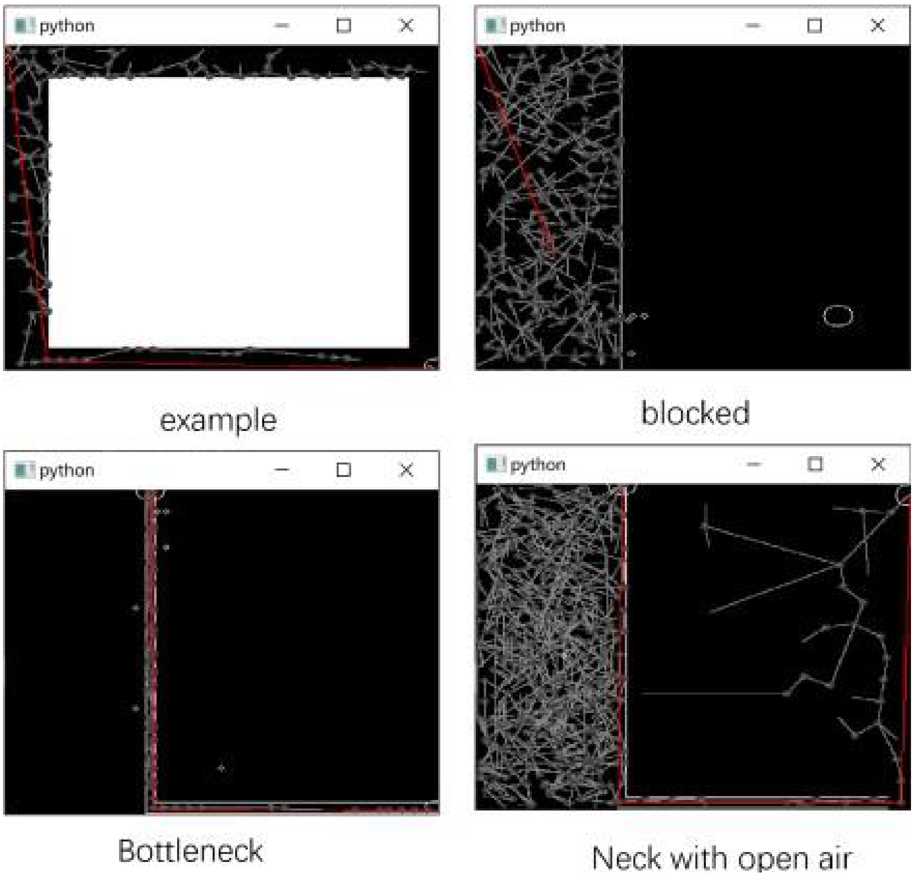
Finished Steps: ___________ /4 ___________

4 Task 4: Path planning (30%)

Please think about how to modify the parameters of RRT to solve the problems that RRT is difficult to deal with.
When finished with each case, do not forget to take a photo; Because RRT is an algorithm based on random selection, you may not get the same result when you submit it. The modification of parameters could only ensure RRT can complete this task with a high probability:

1. Read the code in “`RRT_Test.py`” in `lab8_project/`
2. Run “`RRT_Test.py`”
3. Finish the TODOs in “`RRT_test.py`” case 1 to 3. Save the outputs of 4 cases.

After you finish this task, please show it to the TA.



example

blocked

Bottleneck

Neck with open air

Figure 2: You should have got something like this for your Task 4.

Checked by TA: ______________________________

Finished Time: ______________________________