

# CS3630 Project 1: Trash Sorting Robot (Spring 2022)

Dev TAs: Junyan Mao, Andrew Messing

January 24, 2022

## Contents

<b>1. Introduction &amp; Setup</b>	<b>2</b>
1.1. Introduction . . . . .	2
1.2. Environment Setup . . . . .	2
1.3. Publisher-Subscriber Model . . . . .	3
1.4. Folder Structure . . . . .	4
<b>2. Project - Coding</b>	<b>5</b>
2.1. Modeling the World State ( <a href="#">Book Section 2.1</a> ) . . . . .	5
2.2. Actions for Sorting Trash ( <a href="#">Book Section 2.2</a> ) . . . . .	5
2.3. Sensors for Sorting Trash ( <a href="#">Book Section 2.3</a> ) . . . . .	5
2.4. Perception ( <a href="#">Book Section 2.4</a> ) . . . . .	5
2.5. Decision Theory ( <a href="#">Book Section 2.5</a> ) . . . . .	6
2.6. Extra Credit: Learning ( <a href="#">Book Section 2.6</a> ) . . . . .	6
<b>3. Project - Report</b>	<b>6</b>
<b>4. Logistics &amp; Grading</b>	<b>6</b>
4.1. Due Date . . . . .	6
4.2. Deliverables . . . . .	6
4.3. Grading . . . . .	6
4.4. Collaboration Policy . . . . .	7
4.5. Useful Links . . . . .	7
<b>Appendices</b>	<b>7</b>
<b>A. Linux Virtual Machine Setup Guide</b>	<b>7</b>

# 1. Introduction & Setup

## 1.1. Introduction

Welcome to your first project in CS3630 (Spring 2022)!

Some of you may have heard that this course has a lab portion that deals with “actual” robots, but given the high enrollment of this semester and the current pandemic, we will not have the bandwidth for that. Instead, we will be building robot applications in a simulated environment with Robot Operating System (ROS) using Python. ROS is a set of software libraries and tools for building robot applications. There are multiple versions of ROS available. In this course, we will be using the latest ROS distribution **ROS2 Galactic** throughout the semester. The ROS official website (linked above) has very detailed documentation and a lot of tutorials covering different concepts. Feel free to refer to them at any time.

In this project, we will be building a (simulated) trash sorting robot as illustrated in the [textbook](#) for this course. In this scenario, the robot tries to sort trash of some pre-determined categories into corresponding bins. Please refer to [Section 2](#) of the book for a more detailed description of the scenario. You will also have to complete a report that assesses your understanding of the concepts discussed in the book section.

## 1.2. Environment Setup

**IMPORTANT NOTE:** Unfortunately, there is not a windows version of GTSAM at this time. As such, if you are using a windows machine to work on your project, please use Windows Subsystem for Linux ([guide here](#)), or follow the virtual machine tutorial in the appendix of the instructions.

In this section, we will be installing ROS2 Galactic through [RoboStack](#). RoboStack is a bundling of the ROS by Open Robotics for Linux, Mac and Windows using the Conda package manager. If you haven't already done so, go to [this link](#) and install Miniconda for your system. (Note: If you are on Windows, please install Miniconda in a path that has **NO space** in its name.) Once it's installed run the following commands in a terminal (terminal refers to the terminal application on macOS and Linux, on Windows it should be the Anaconda Powershell application):

```
# Install mamba
conda install mamba -c conda-forge
# Create a new environment
mamba create -n ros_env python=3.8
# Activate the environment
conda activate ros_env
# Add the conda-forge channel to environment configuration
conda config --env --add channels conda-forge
# Add channel for ROS2 Galactic
conda config --env --add channels robostack-experimental
# Use strict channel priority
conda config --env --set channel_priority strict
# Install ROS2 Galactic
mamba install ros-galactic-desktop
# Additionally, install gtbook (this also installs gtsam)
pip install -U gtbook
# Re-activate the environment
conda deactivate
conda activate ros_env
```

**IMPORTANT NOTE:** Make sure that the installed version of gtbook is 0.0.13 or greater and the installed version of gtsam is 4.2a3 or greater.

To check if your ROS2 installation works, open two terminals, activate the `ros_env` environment by running `conda activate ros_env`, run the following code in one terminal:

```
ros2 run demo_nodes_cpp talker
```

and run the following code in another:

```
ros2 run demo_nodes_cpp listener
```

If your installation is correct, you should be able to see a talker publishing a message and a listener receiving a message as in the following picture. Note that this is an example of the publisher-subscriber model that will be discussed in the next section.

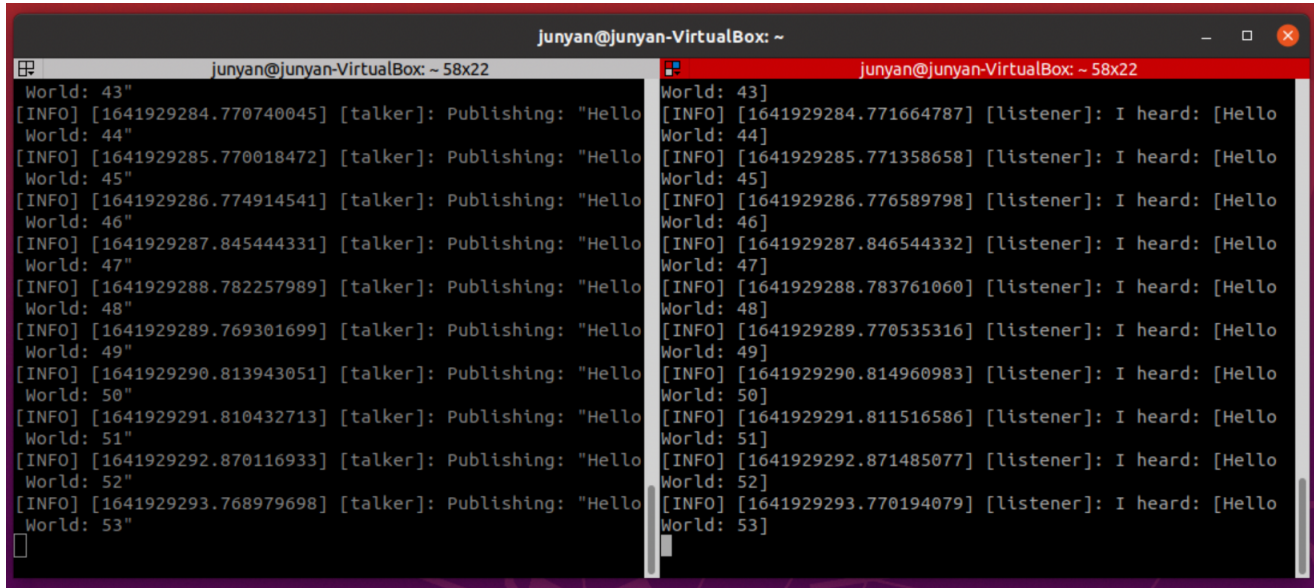


Figure 1: Talker-Listener example

We are expecting many different issues with this installation on macOS and Windows due to different system configurations that everyone has. If you run into any error and are able to solve it, we encourage you to share your debugging experience on Piazza. We will also try our best to answer these setup questions. If you have no luck getting ROS2 to run on your system, don't worry, please refer to Appendix A for the Linux virtual machine setup guide.

### 1.3. Publisher-Subscriber Model

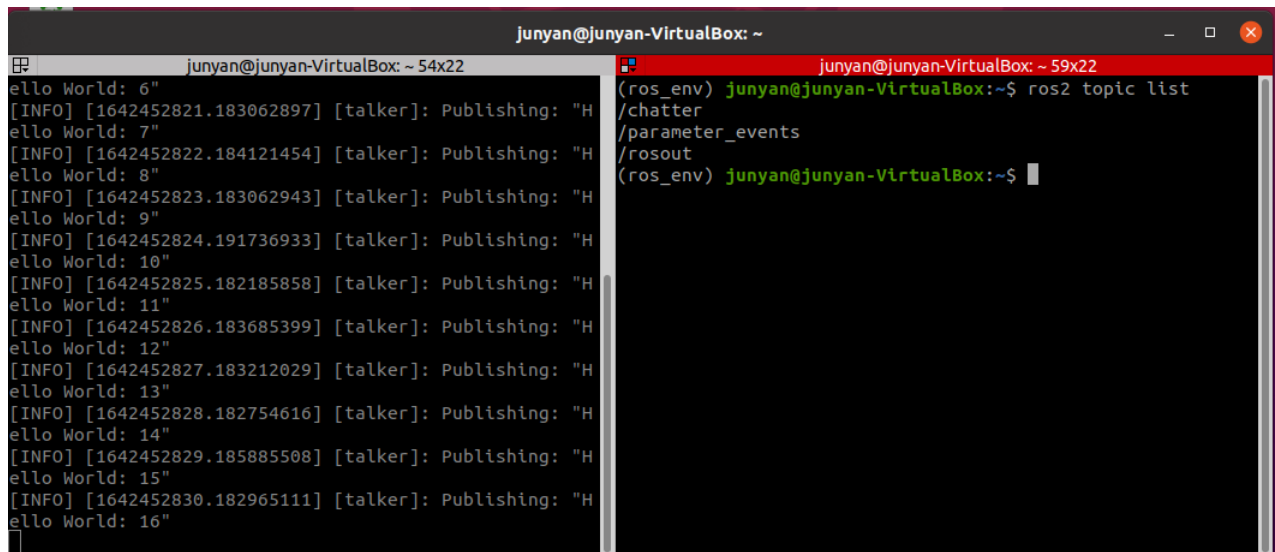
There are many ways for robots to communicate with the environment or with each other. In this project, we will be diving into the publisher-subscriber model. To help you better understand this concept, we will introduce a few terminologies:

- Node - A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another.
- Message - Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields.
- Topic - Topics are named buses over which nodes exchange messages. Note that each topic is strongly typed by the ROS message used to publish it and nodes can only receive messages with a matching type. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes that are interested in data **subscribe** to the relevant

topic (subscriber); nodes that generate data **publish** to the relevant topic (publisher). There can be multiple publishers and subscribers to a topic.

- Publisher - A node that creates and sends messages to a topic(s).cx
- Subscriber - A node with a subscription to a topic(s) to receive messages from it.

Let's look back at the publisher-subscriber model. As the name suggests, there is usually (at least) 1 publisher and (at least) 1 subscriber in the model. There are multiple publisher-subscriber relationships in this project. For example, there is a publisher for the conductivity sensor in **world.py** and a subscriber for this in **brain.py**. There is also a publisher for the final sorting decision in **brain.py**, while its subscriber lives in **world.py**. To see a list of all active topics, you can start your publisher and run the command `ros2 topic list`. Note that in the image below, the **demo\_talker** node is publishing to the **chatter** topic.



The image shows two terminal windows from a Junyan VirtualBox. The left window, titled 'junyan@junyan-VirtualBox: ~ 54x22', displays the output of a node publishing to the 'chatter' topic. It shows a series of 'ello World: 6' through 'ello World: 16' messages, each preceded by an INFO log line indicating the publisher's address and the topic name. The right window, titled 'junyan@junyan-VirtualBox: ~ 59x22', shows the command 'ros2 topic list' being executed, which returns a list of active topics: '/chatter', '/parameter\_events', and '/rosout'.

```
junyan@junyan-VirtualBox: ~ 54x22
ello World: 6"
[INFO] [1642452821.183062897] [talker]: Publishing: "H
ello World: 7"
[INFO] [1642452822.184121454] [talker]: Publishing: "H
ello World: 8"
[INFO] [1642452823.183062943] [talker]: Publishing: "H
ello World: 9"
[INFO] [1642452824.191736933] [talker]: Publishing: "H
ello World: 10"
[INFO] [1642452825.182185858] [talker]: Publishing: "H
ello World: 11"
[INFO] [1642452826.183685399] [talker]: Publishing: "H
ello World: 12"
[INFO] [1642452827.183212029] [talker]: Publishing: "H
ello World: 13"
[INFO] [1642452828.182754616] [talker]: Publishing: "H
ello World: 14"
[INFO] [1642452829.185885508] [talker]: Publishing: "H
ello World: 15"
[INFO] [1642452830.182965111] [talker]: Publishing: "H
ello World: 16"

junyan@junyan-VirtualBox: ~ 59x22
(ros_env) junyan@junyan-VirtualBox:~$ ros2 topic list
/chatter
/parameter_events
/rosout
(ros_env) junyan@junyan-VirtualBox:~$
```

Figure 2: Active topic list

## 1.4. Folder Structure

In this folder, along with this pdf you will find these 5 files:

- **brain.py** - This file contains all the robot functions, including perception of the information received from world and decision making. You can simply start the robot by running `python brain.py`.
- **world.py** - This file contains all the world functions, including sampling the trash category and sampling sensor readings with probability. You can simply start the world by running `python world.py`.
- **utils.py** - This file contains some universal variables and functions that we will be using across the world and the brain.
- **tests.py** - This file contains the unit tests for this project.
- **collect\_submission.py** - This file contains a script that will create a zip file that you can submitted to Gradescope.
- **project1\_report.pptx** - This file contains the report template which you need to fill out.

## 2. Project - Coding

There are three ways that you can test your implementation:

1. **Running the project:** Running the project involves running both `brain.py`, which simulates the brain of a robot that is sorting trash, and `world.py`, which simulates the world in this scenario. Both of these files contain multiple TODO functions that must be completed for the project to run. Both of these files will be running simultaneously, so you will need multiple terminal tabs or windows. Make sure that each tab or window has activated the conda environment (`conda activate ros_env`). In `brain.py` you can set which perception algorithm you wish the robot to use. In `world.py`, you can set the number of trash to simulate. The world will then provide with a score based on the cost table from the book. A lower score is better.

**IMPORTANT NOTE: When you are running `brain.py` and `world.py`, make sure you start `brain.py` first.**

2. **Unittests:** Some public unit tests are provided in the `tests.py` file. You can test each part of your implementation with these test cases by:

```
python tests.py TestProject1.test_#####
```

Note: passing all of the local tests neither means your code is free of bugs nor guarantees that you will receive full credit for the coding section. Your code will be graded by a Gradescope Autograder (see below for more details). There will be additional tests on Gradescope which are not present in your local unit tests.

3. **Gradescope Autograder:** You may also submit your code as specified in Section 3.1 for testing. Gradescope will only reveal the results of the public tests. Your final grade for the coding portion of this assignment are based on both the public and hidden private tests. We do not recommend using Gradescope as your primary testing method during development because private test cases will NOT be available to you at any time.

**IMPORTANT NOTE: Please use the variables provided for the results of each of the TODOs.**

### 2.1. Modeling the World State ([Book Section 2.1](#))

- Functions to complete: **TODO 1** and **TODO 2** in `utils.py`, **TODO 3** in `world.py`
- Objective: Representing the prior probabilities of the trash categories and simulate it by sampling

### 2.2. Actions for Sorting Trash ([Book Section 2.2](#))

- Functions to complete: **TODO 4** in `utils.py`
- Objective: Representing actions and their corresponding costs

### 2.3. Sensors for Sorting Trash ([Book Section 2.3](#))

- Functions to complete: **TODO 5-7** in `utils.py`, **TODO 8-10** in `world.py`
- Objective: Representing conditional probabilities of sensors and simulate them by sampling

### 2.4. Perception ([Book Section 2.4](#))

- Functions to complete: **TODO 11-15** in `brain.py`
- Objective: Calculating likelihoods using different methods given the observations from the world

## 2.5. Decision Theory ([Book Section 2.5](#))

- Functions to complete: **TODO 16** in `brain.py`
- Objective: Incorporating the cost table with the perception to reach a final sorting decision

## 2.6. Extra Credit: Learning ([Book Section 2.6](#))

A Gaussian distribution, also known as a normal distribution, is an inappropriate distribution to represent the weight of an item. This is because it has an infinite range and therefore sampling from it can produce a negative number, while an item cannot have a negative weight. A more commonly used distribution used to represent weight is the [log-normal distribution](#) which can only contain positive real values. The book explains how to fit a gaussian distribution to a set of data. For extra credit, we would like you to implement a function

- Functions to complete: **TODO 17** in `utils.py`
- Objective: Fit a Log-Normal Distribution to a set of data
- Hint: There is an estimation of parameters section on the wikipedia article.

## 3. Project - Report

In this project, you will also have to answer a few questions (listed in `project1_report_empty.pdf`) that test your understanding of probability and decision theory. If a question involves calculation, you have to show all the steps used to arrive at your answer. Providing the answer alone will result in partial credit.

## 4. Logistics & Grading

### 4.1. Due Date

Project 1 is due on **Friday, 02/04/2022 11:59PM Easter Time**, with late submission possible until **Monday, 02/07/2022 11:59PM Eastern Time**.

### 4.2. Deliverables

Deliverables are submitted on Gradescope. Each student must individually submit assignments. Deliverables are the following files:

- `submission.zip`: Simply run `python collect_submission.py` and upload the result zip file to Gradescope for “Project 1 - Code”.
- Project 1 - Report: Reports are administered through GradeScope assignments. The assignment is not timed, you will have until the deadline to finish the report. Fill out the corresponding regions with your answers.

### 4.3. Grading

Project 1 is worth 100 points in total. Here is the breakdown of each section’s value.

- Code - 50pts (5pts extra credit possible)
- Report - 50pts

## 4.4. Collaboration Policy

All projects in this course this semester are individual projects, which means no collaboration is allowed. However, you can discuss the projects and the reflection questions at a high level with your classmates or the TAs. You can also ask questions on Piazza or go to office hours for help. All the code and all the answers to the reflection questions must be your own. However, feel free to refer to the example code in the official ROS2 guide or the code in the robotics book in your project.

## 4.5. Useful Links

- [ROS2 Galactic official website](#)
- [rclpy \(ROS Client Library\) Documentation](#)

# Appendices

## A. Linux Virtual Machine Setup Guide

Setting up ROS2 on MacOS and Windows may result in various errors that are hard to debug. To solve this issue, we provide a setup guide for a Ubuntu 20.04 LTS virtual machine via VirtualBox, which provides us a fresh and unified environment for this course. If you ran into problems in the previous setup, please refer to this guide. Here are the steps:

1. Downloading:
  - a) Download [VirtualBox](#) and follow the installation guide
  - b) Download [Ubuntu 20.04 LTS](#) and put it somewhere you can easily locate
2. Starting up the VM:
  - a) Adding a new VM:
    - i. Start VirtualBox, click on the “**New**” button, and fill out fields according to the following picture. The “**Machine Folder**” path in the picture is specific to a TA’s laptop, you can choose any folder on your laptop.

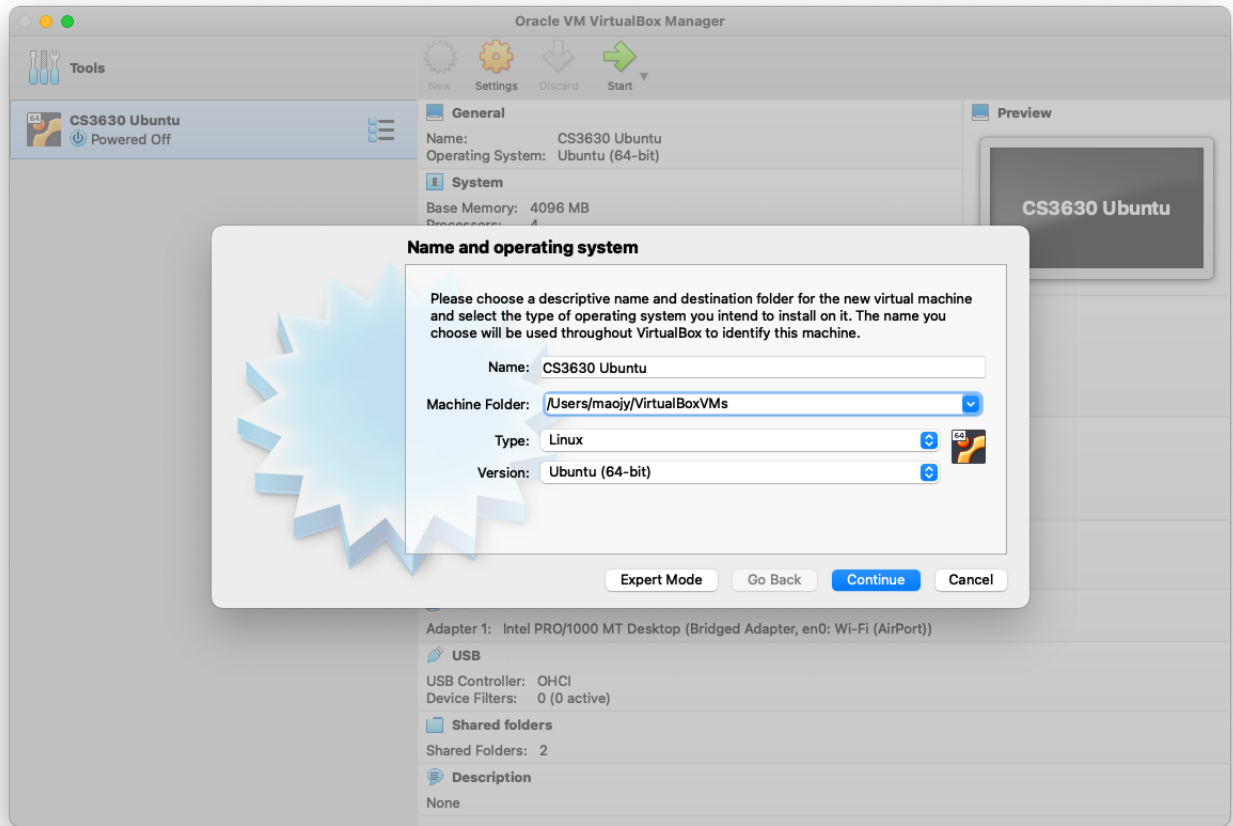


Figure 3: Name and operating system of the VM

- ii. On the next screen, we recommend **4096 MB** as memory size for the virtual machine. Feel free to set a higher memory size if your machine has more capacity.

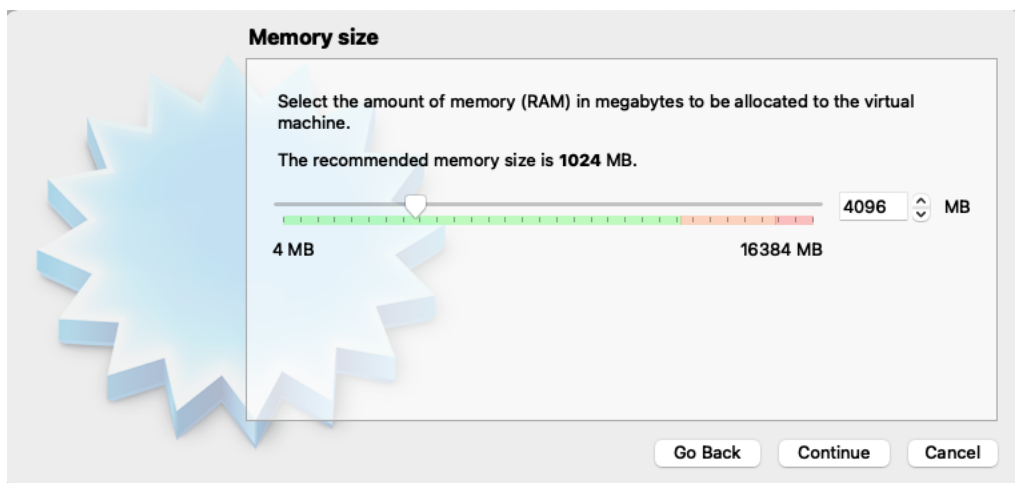


Figure 4: Memory size of the VM

- iii. On the next screen, choose “**Create a virtual hard disk now**”



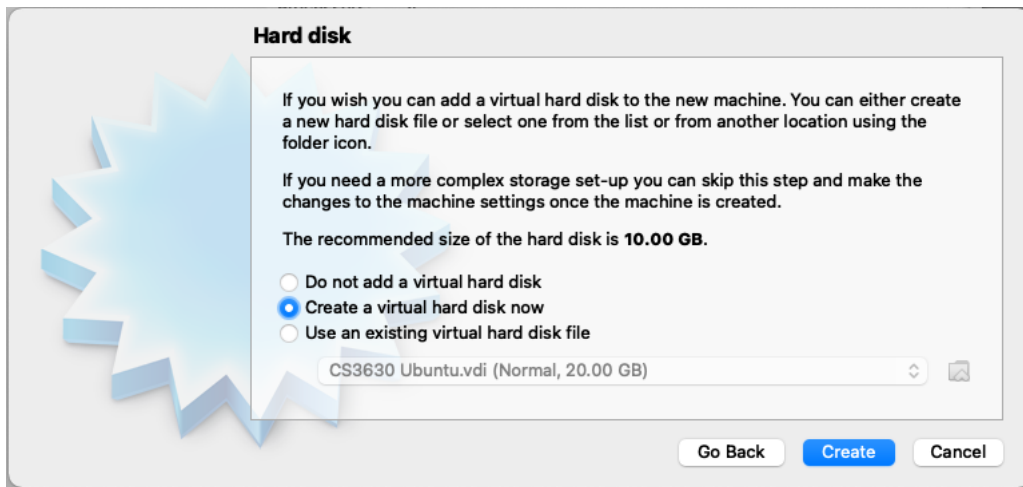


Figure 5: Creating a virtual hard disk

- iv. On the next screen, keep the default choice “**VDI(VirtualBox Disk Image)**”

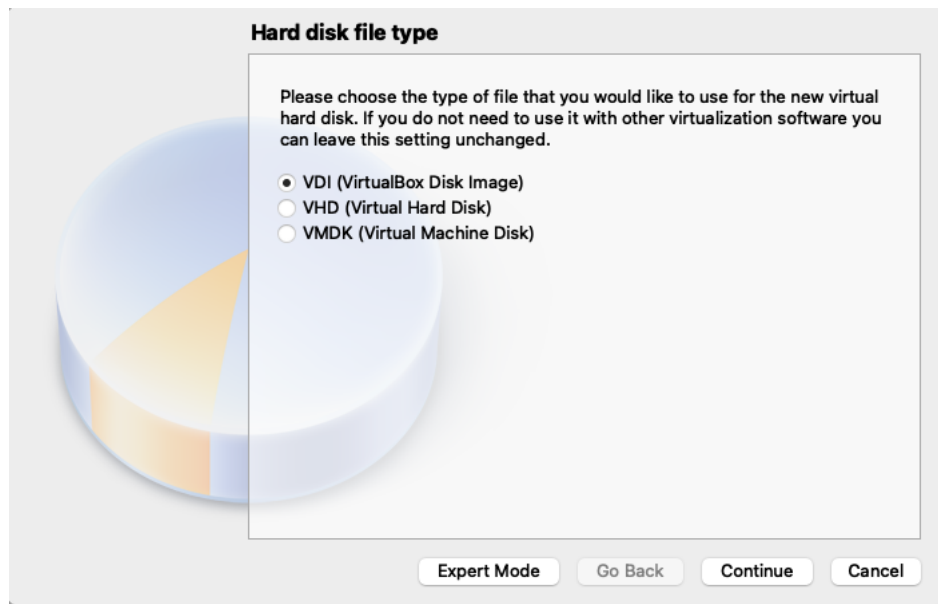


Figure 6: Hard disk file type

- v. On the next screen, choose “**Dynamically allocated**”

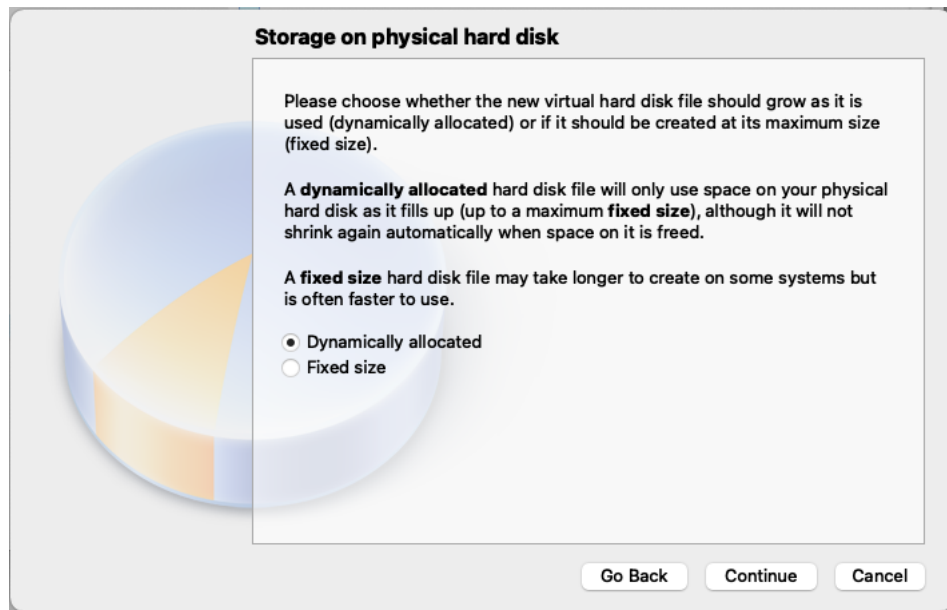


Figure 7: Storage on physical hard disk

- vi. On the next screen, we recommend a size limit of at least **15 GB** (or larger). Note that this does not mean the VM will take up 15 GB of your hard disk immediately. Rather, it's a size limit of what the VM can write to your hard disk. Please keep the file location as default and hit the **“Create”** button.

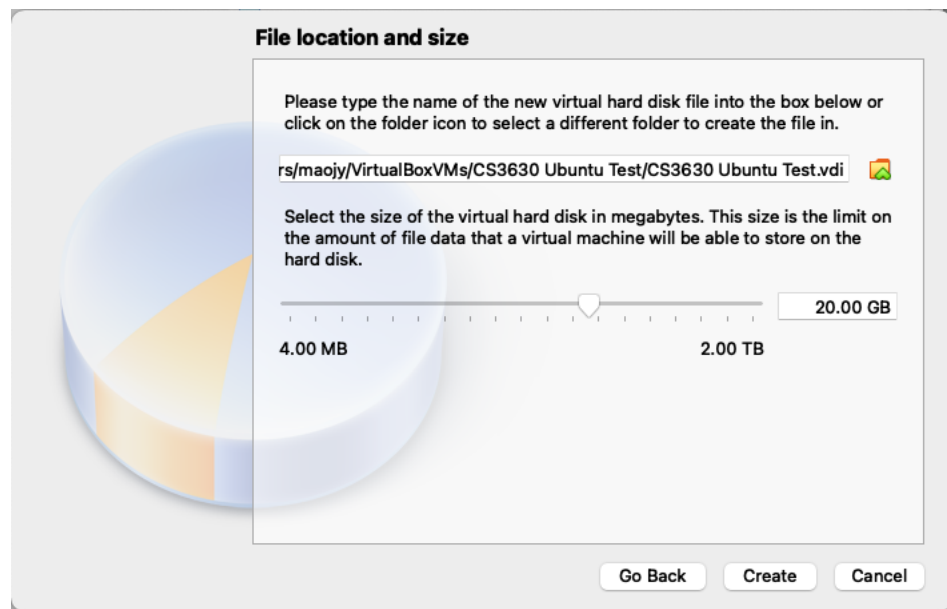


Figure 8: File location and size

- b) Configuring the VM - there're some things we need to do for the VM to function correctly
  - i. Go to **Settings** → **System** → **Processor** and change the number of processors to at least 2 (4 is recommended), so that your VM has enough computing power, then click **OK**.

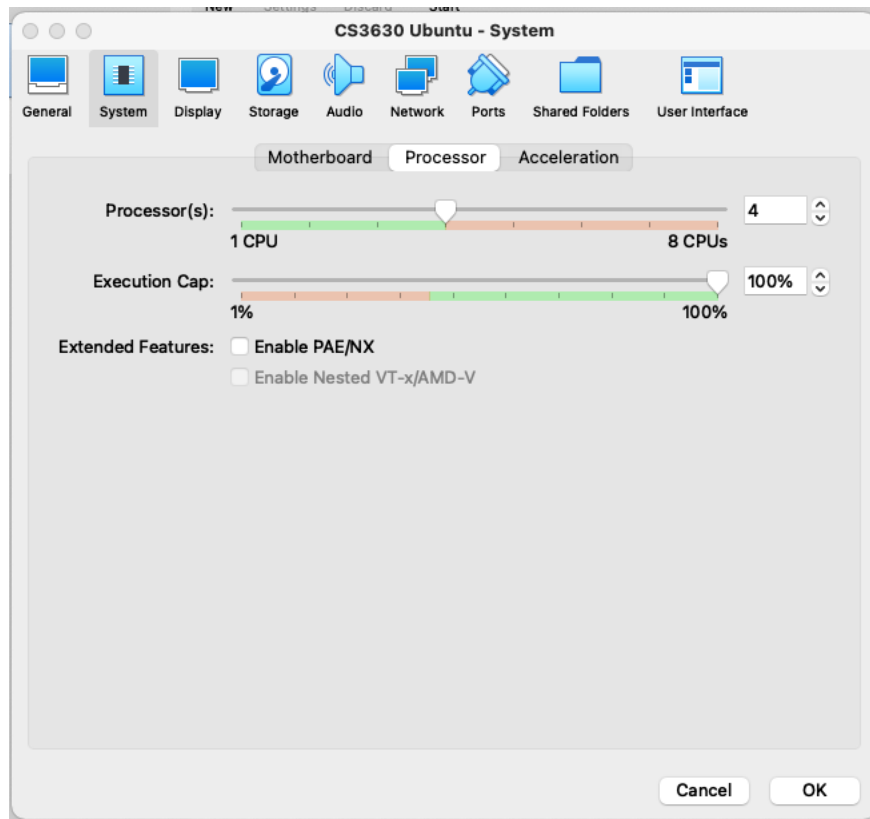


Figure 9: Processor count

- ii. Go to **Settings** → **Network** → **Adapter 1** and change it to **Attached to: NAT** so that your VM has access to the Internet.

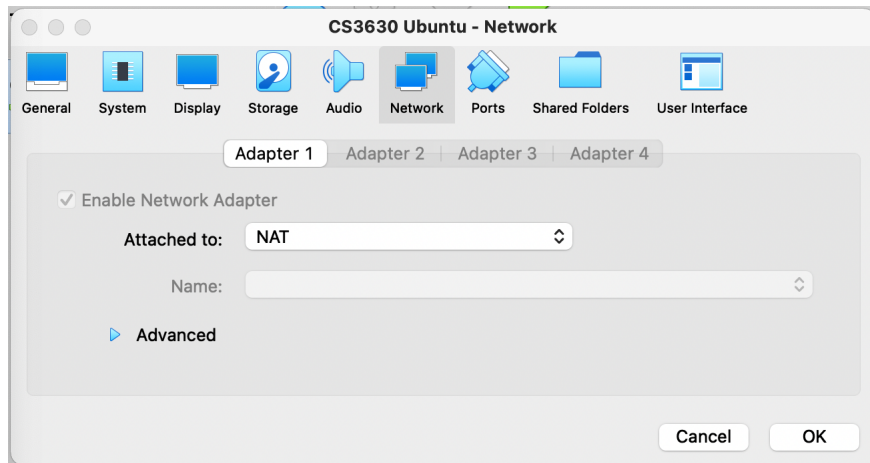


Figure 10: VM network adapter

- iii. Go to **Settings** → **Storage**, click on the **Empty** under **Controller: IDE**, then click on the blue dot next to **IDE Secondary Device 0**, select **Choose a disk file...**, locate the Ubuntu disk image you downloaded from Step 1, and finally click **OK**. Now you are ready to start the virtual machine for the first time.

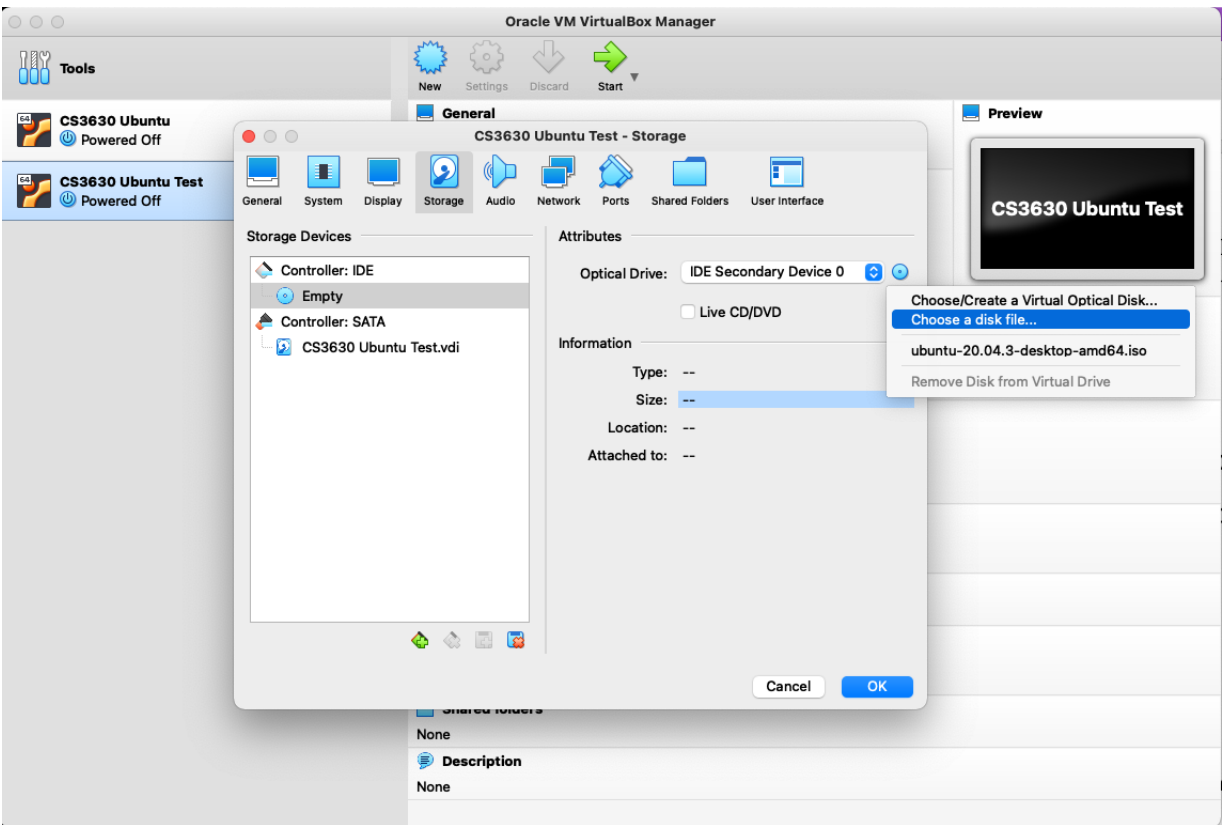


Figure 11: Choose disk file

- c) Starting the VM:
- i. Click on the **Start** button, and wait for your VM to boot.
  - ii. On the first screen you see, click **Install Ubuntu**.

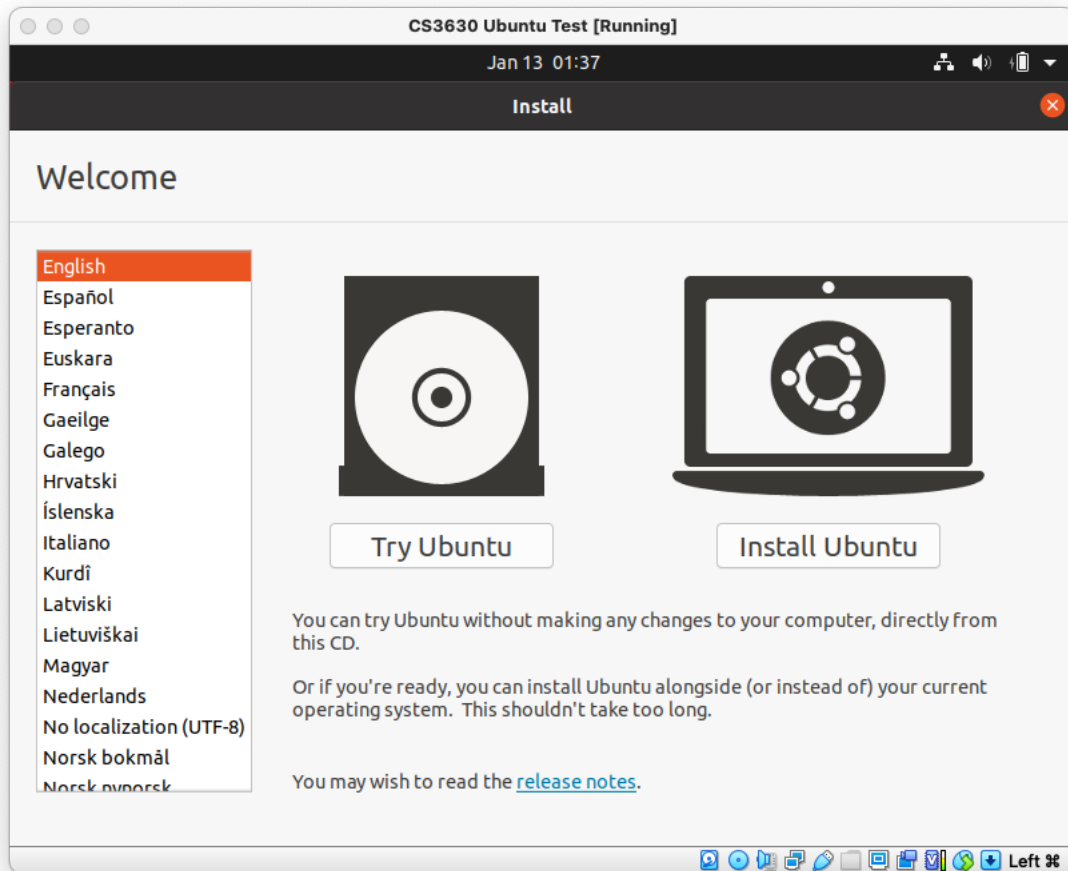


Figure 12: Install Ubuntu

- iii. On the next screen, choose your keyboard layout.

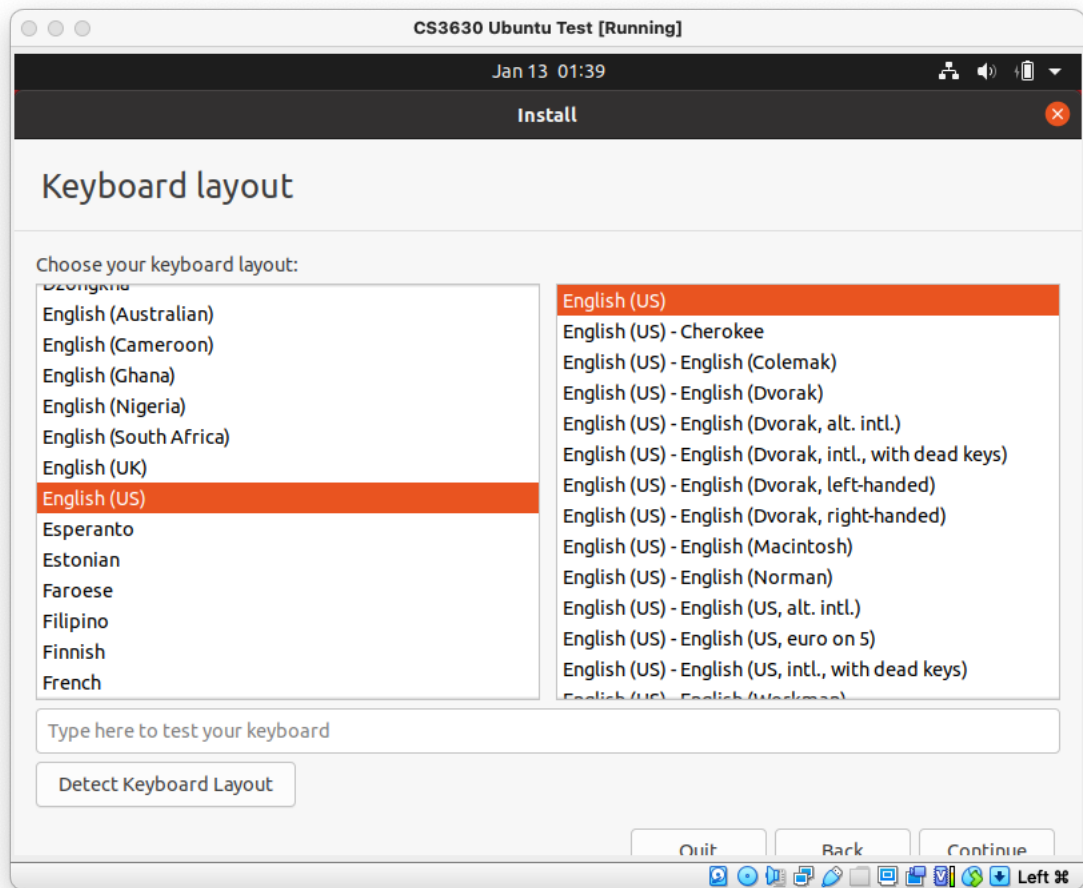


Figure 13: Keyboard layout

iv. On the next screen, choose **Minimal installation**

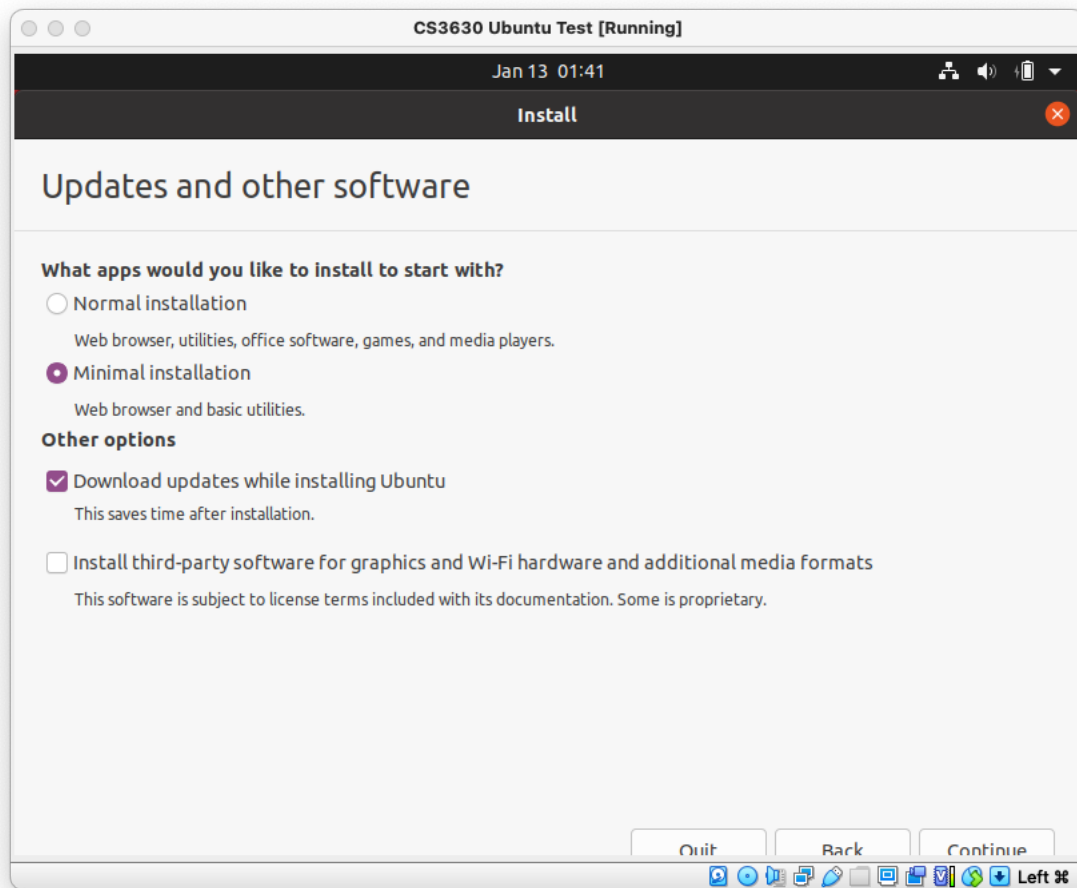


Figure 14: Updates and other software

- v. On the next screen, choose **Erase disk and install Ubuntu**, click **Install Now**, and then **Continue**.

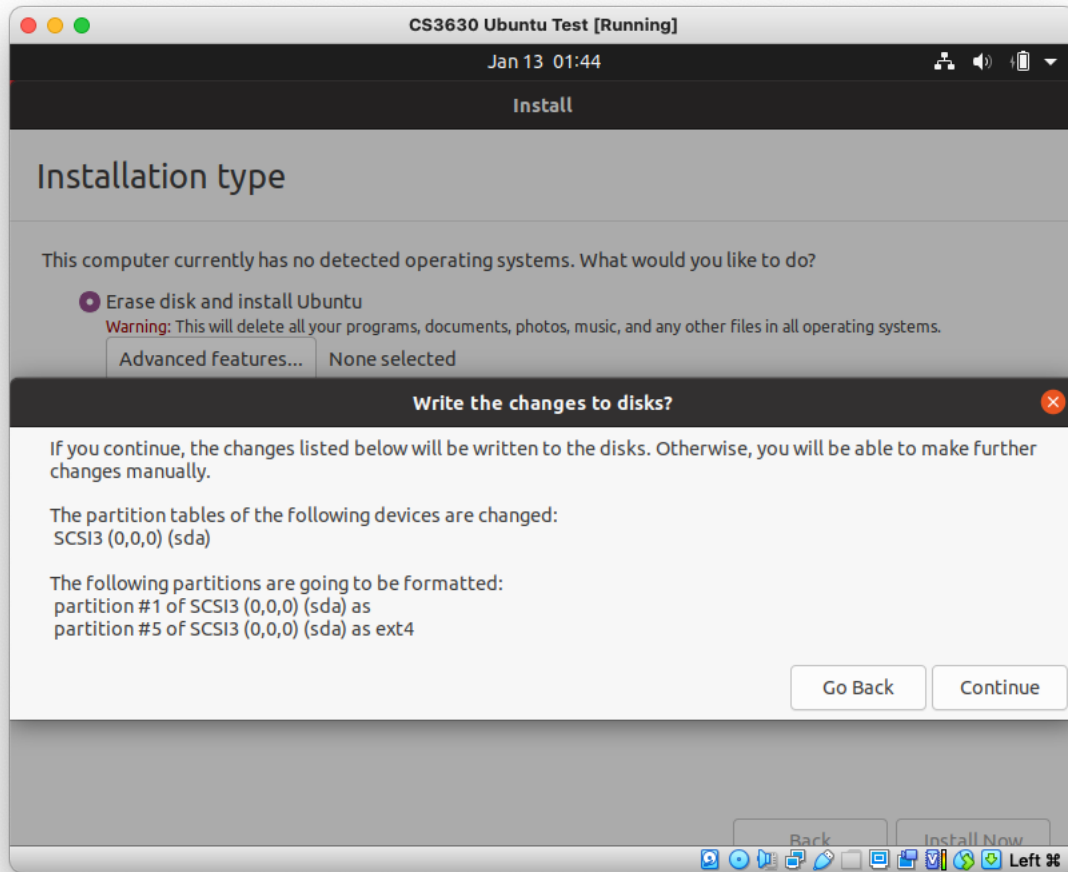


Figure 15: Installation type

- vi. On the next screen, choose your location accordingly.
- vii. On the next screen, fill in your name, computer's name, username, and a password. We recommend a shorter password for easier access to your VM. You can click **Continue** and the installation will start. This may take a while.
- viii. If you see this screen in the end, you have successfully set up your Ubuntu VM!



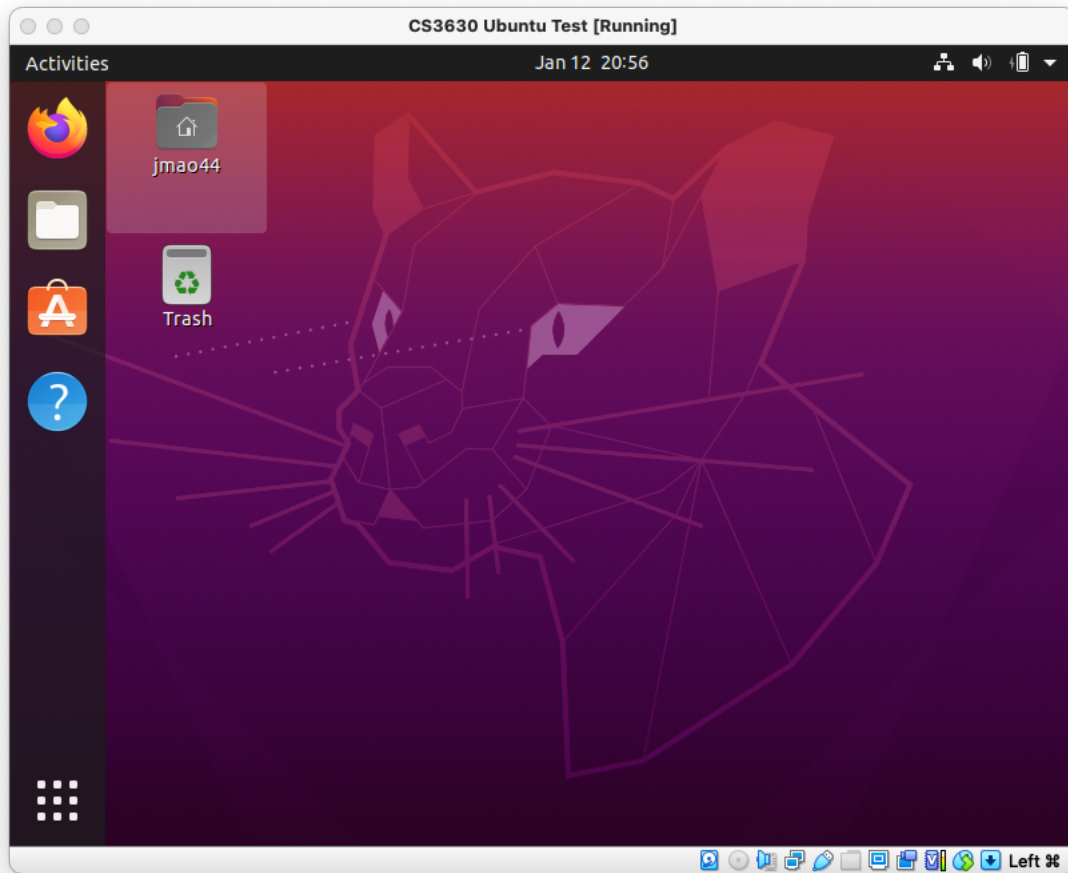


Figure 16: Ubuntu starting screen

3. Some useful utilities and tools:

- a) Screen resizing. You may have noticed that the screen of your VM will not change however you resize the outer window. To fix this, go to **Devices** → **Insert Guest Additions CD Image...**, and click on **Run** when you see the pop-up screen. Restart (power off and start) the VM when you see a completion message. You should be able to resize your window now. (Note: if you still cannot resize even after this step, eject the CD image by right-click on it in the left menu bar, and follow this step from the beginning again.)
- b) Shared clipboard. To share the content on the clipboard between the host machine and the VM, go to **Devices** → **Shared Clipboard** and select **Bidirectional**.
- c) Shared folder (highly recommended). To keep the VM as lightweight as possible, we can edit code in the host machine and only run code in the VM. We can set up a shared folder between the two by going to **Devices** → **Shared Folders** → **Shared Folder Settings...**

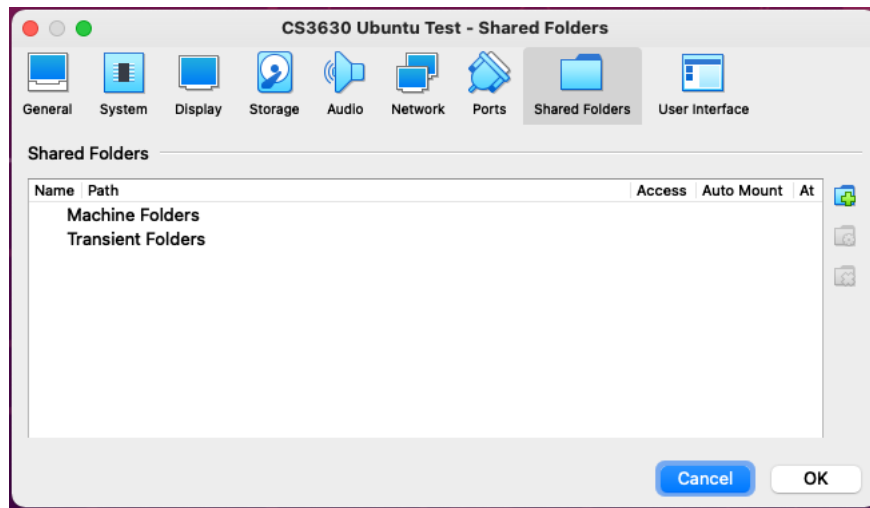


Figure 17: Shared folder setup

Click the **add** button on the right and select the folder you want to share (we suggest sharing the entire folder for all your 3630 stuff). Also make sure that **Auto-mount** is selected.

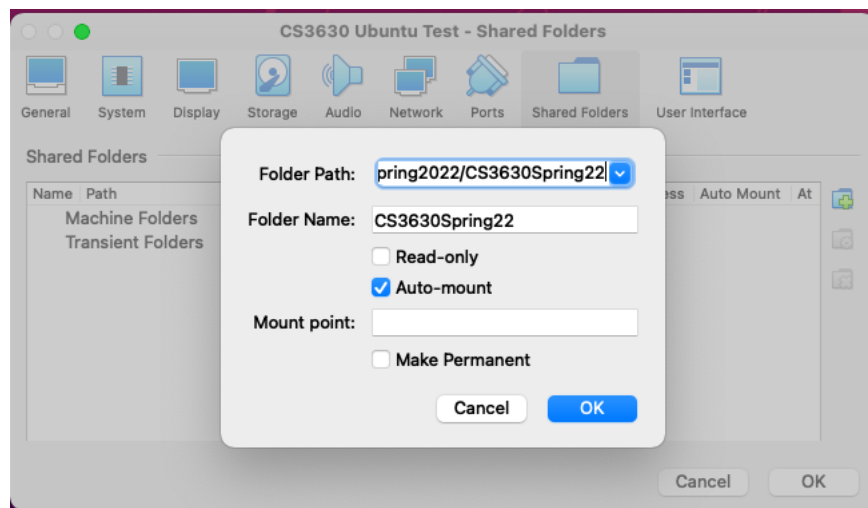
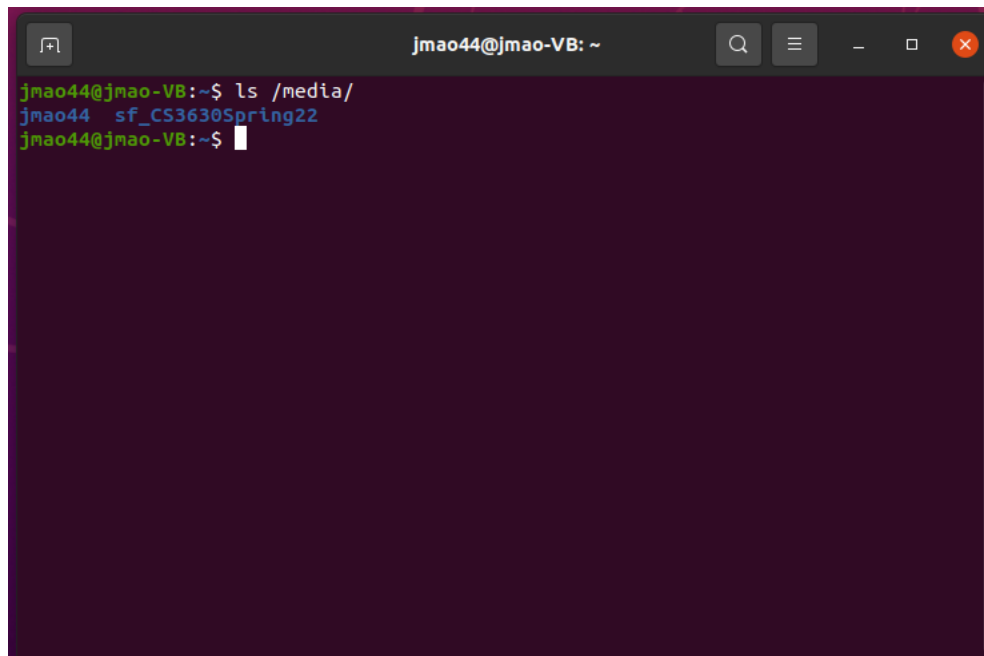


Figure 18: Shared folder options

Now it's a good time to restart your VM so that all these changes can be applied. All the shared folders are stored in `/media/` on the VM. For example, my shared folder appears as `/media/sf_CS3630Spring22` on the VM, and it's corresponding to the `CS3630Spring22` folder on my host machine. The contents are synced in real-time, which means that you can edit code on your favorite editor on your host machine, and run the same piece of code in your VM without having to refresh or anything.

A terminal window with a dark background and light green text. The window title is 'jmao44@jmao-VB: ~'. The prompt is 'jmao44@jmao-VB:~\$'. The command 'ls /media/' has been entered, and the output 'sf\_CS3630Spring22' is displayed on the next line. The prompt 'jmao44@jmao-VB:~\$' is shown again on the third line.

```
jmao44@jmao-VB:~$ ls /media/
jmao44  sf_CS3630Spring22
jmao44@jmao-VB:~$
```

Figure 19: Shared folder path

#### 4. Project setup:

- a) Miniconda. We are using `conda` as our Python environment manager in this course. On your VM, go to <https://docs.conda.io/en/latest/miniconda.html> and download the **Miniconda3 Linux 64-bit** installer. It's a shell script and will be downloaded to your `~/Downloads` folder. To start the installer, simply run the following command in a terminal window and follow the instructions:

```
sh ~/Downloads/Miniconda3-latest-Linux-x86_64.sh
```

Type **yes** when you see the following message:

```
jmao44@jmao-VB: ~
pysocks      pkgs/main/linux-64::pysocks-1.7.1-py39h06a4308_0
python       pkgs/main/linux-64::python-3.9.5-h12debd9_4
readline     pkgs/main/linux-64::readline-8.1-h27cfd23_0
requests     pkgs/main/noarch::requests-2.25.1-pyhd3eb1b0_0
ruamel_yaml  pkgs/main/linux-64::ruamel_yaml-0.15.100-py39h27cfd23_0
setuptools   pkgs/main/linux-64::setuptools-52.0.0-py39h06a4308_0
six          pkgs/main/noarch::six-1.16.0-pyhd3eb1b0_0
sqlite       pkgs/main/linux-64::sqlite-3.36.0-hc218d9a_0
tk           pkgs/main/linux-64::tk-8.6.10-hbc83047_0
tqdm         pkgs/main/noarch::tqdm-4.61.2-pyhd3eb1b0_1
tzdata       pkgs/main/noarch::tzdata-2021a-h52ac0ba_0
urllib3      pkgs/main/noarch::urllib3-1.26.6-pyhd3eb1b0_1
wheel        pkgs/main/noarch::wheel-0.36.2-pyhd3eb1b0_0
xz           pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
yaml         pkgs/main/linux-64::yaml-0.2.5-h7b6447c_0
zlib         pkgs/main/linux-64::zlib-1.2.11-h7b6447c_3

Preparing transaction: done
Executing transaction: done
Installation finished.
Do you wish the installer to initialize Miniconda3
by running conda init? [yes|no]
[no] >>>
```

Figure 20: Select yes for conda init

Close the terminal, start a new one, and run the command `conda env list`, you should be able to see the following screen. This means you have successfully installed Miniconda on your VM.

```
jmao44@jmao-VB: ~
(base) jmao44@jmao-VB:~$ conda env list
# conda environments:
#
base                  *  /home/jmao44/miniconda3

(base) jmao44@jmao-VB:~$
```

Figure 21: Result from running `conda env list`

- b) RoboStack. Please follow the guide in Section 1.2 on your VM, and run the two lines of testing code in two separate terminals to verify your ROS2 installation.