Introduction to Ubuntu and the Robot Operating System (ROS)

January 20, 2023

Pooya Poolad

UNIVERSITY OF TORONTO

Virtual Machine and Docker

Getting started with Ubuntu and ROS

Pooya Poolad

Outline

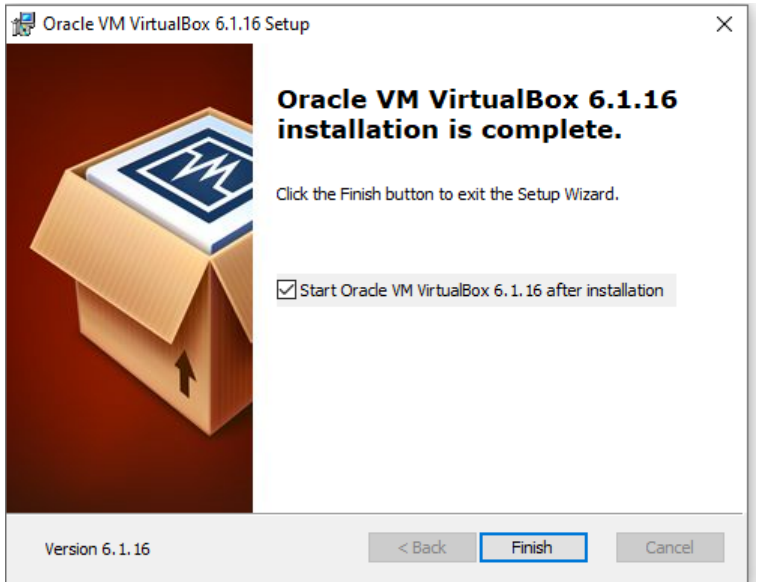
- Installing Ubuntu on a Virtual Machine
- Using Docker on Windows

Installing VM

- Oracle VirtualBox is a Free and powerful software:
  1. Download VirtualBox:
     - https://www.virtualbox.org/wiki/Downloads
  2. Download VirtualBox Extension pack (Make sure it matches your version; 7.0.4 is the latest):
     - https://download.virtualbox.org/virtualbox/7.0.4/Oracle_VM_VirtualBox_Extension_Pack-7.0.4.vbox-extpack

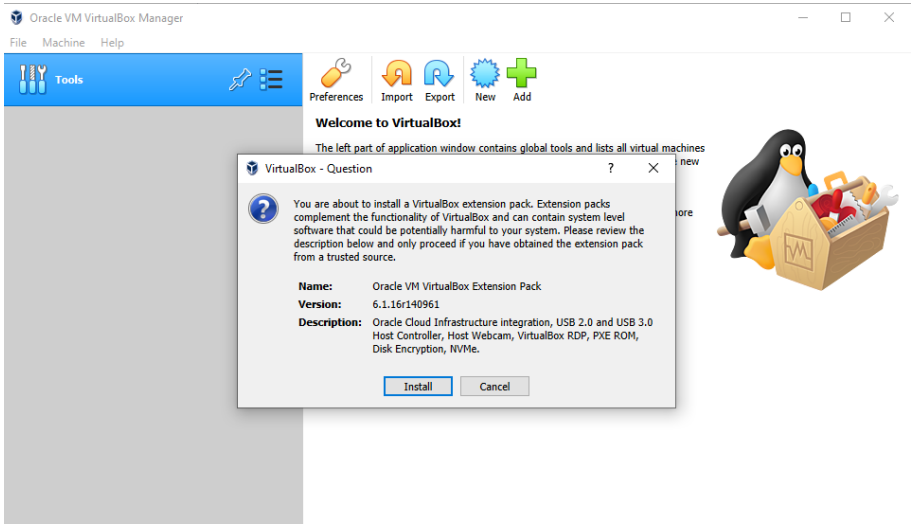
VirtualBox

- Installing VirtualBox is straight forward (Just hit next)
  - Make sure Network driver will be installed



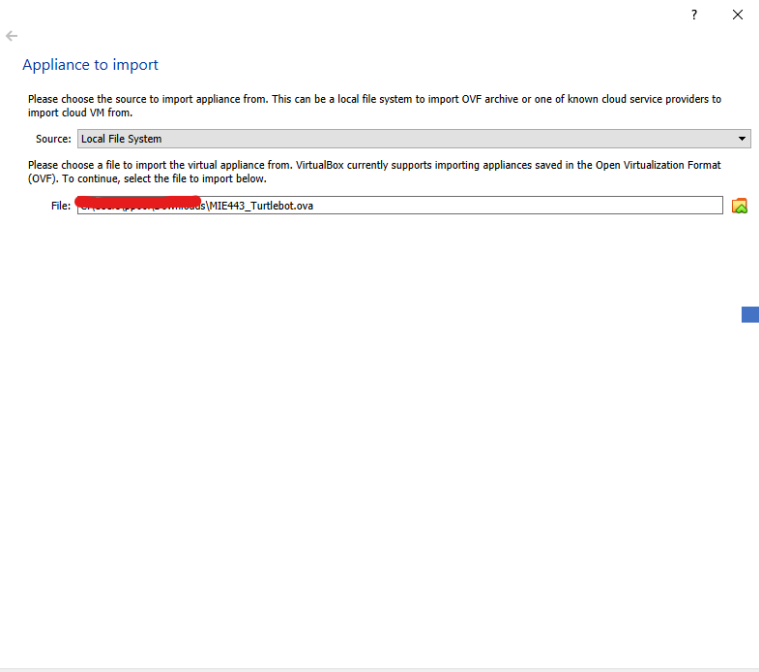
VirtualBox extension

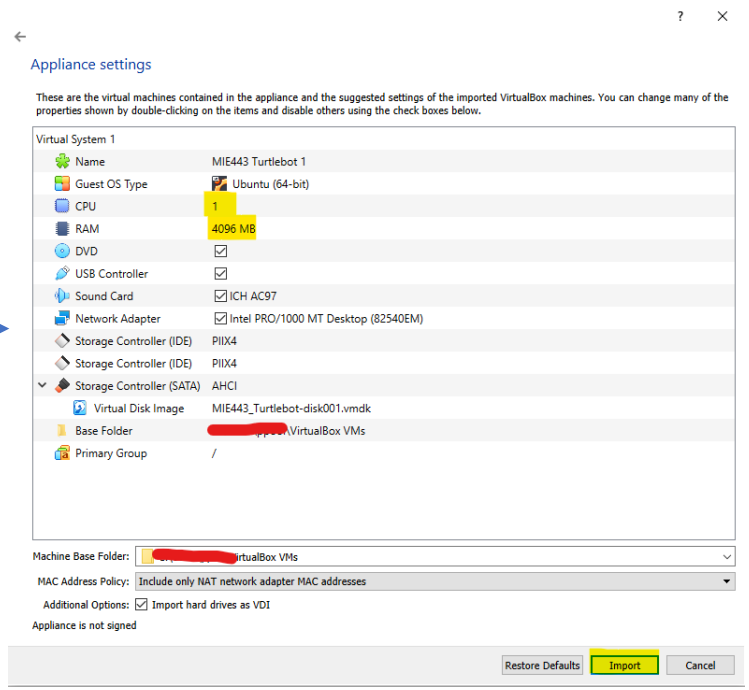
- Double click on the downloaded file
  1. VirtualBox should open
  2. Click on install



Turtlebot Image

- Download Turtlebot image (4.6GB) from:
  - [MIE443_Turtlebot.ova]

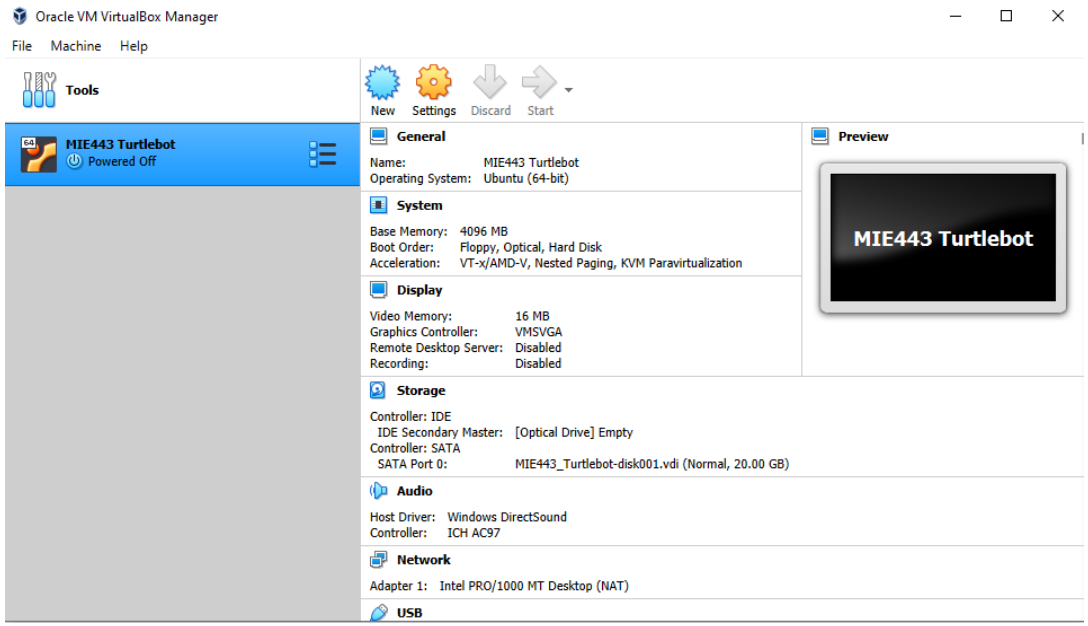




UNIVERSITY OF TORONTO

Imported!

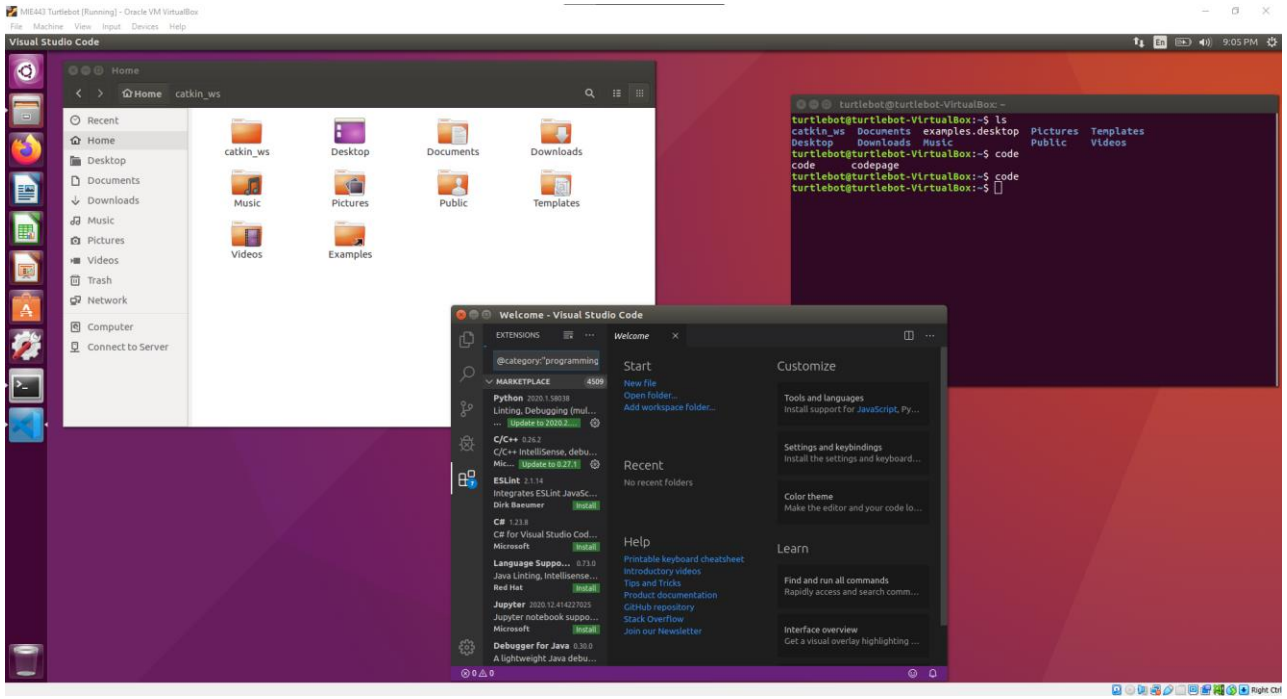
- Right click and start VM.
- Password: turtlebot



UNIVERSITY OF TORONTO

Inside the VM

- Everything is installed.
- You can start exploring right away.



Update Gazebo

- If you ran into issues running simulations on Gazebo, try updating to version 7

```
sudo apt-get install wget
```

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

```
sudo apt update
```

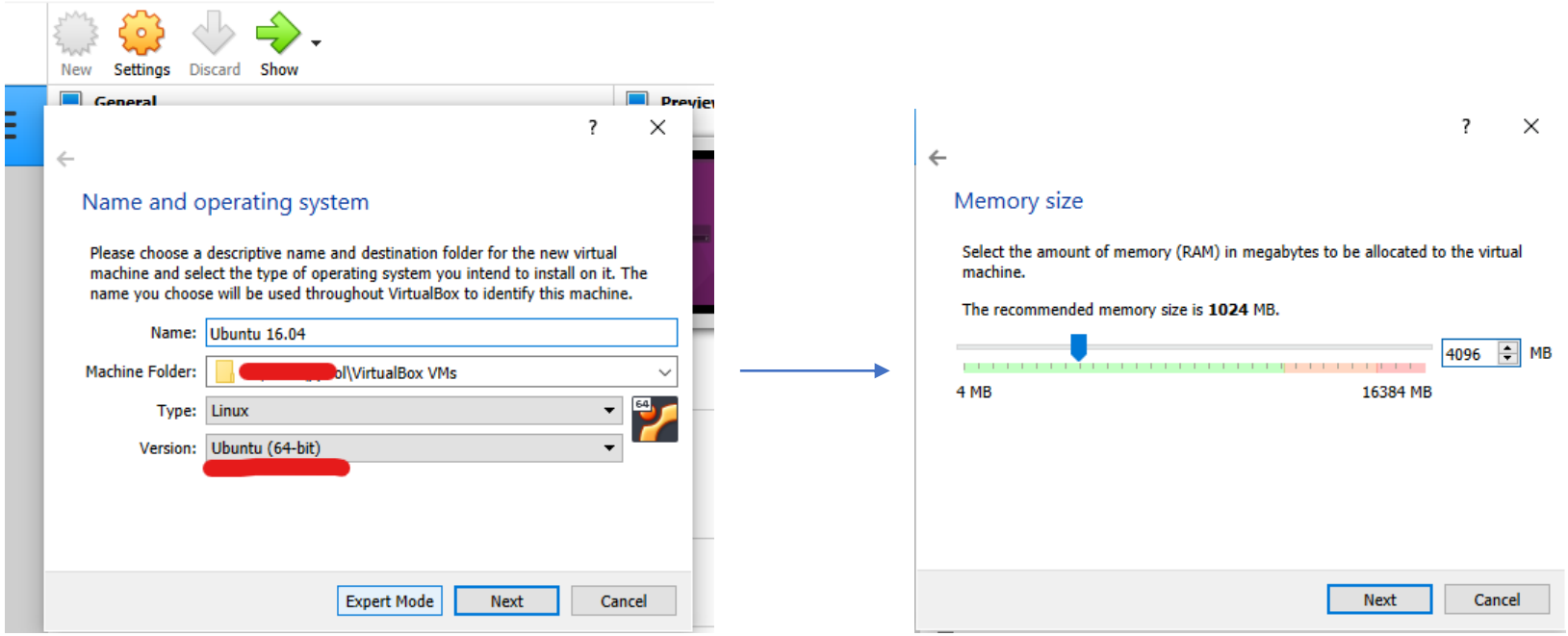
```
sudo apt upgrade
```

Virtual Machine From Scratch

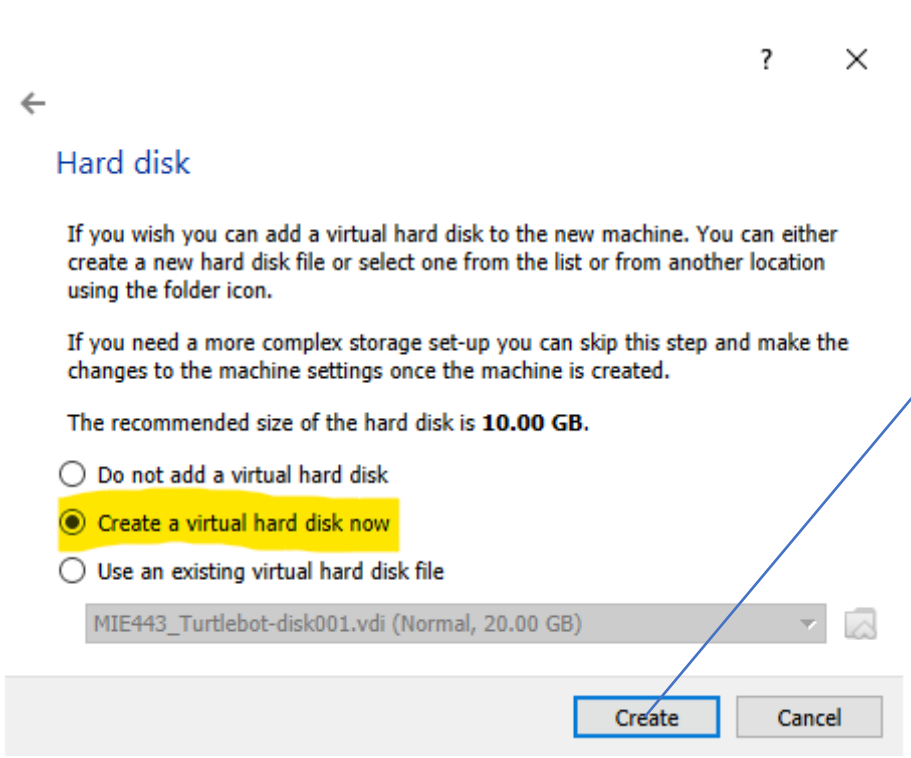
- Alternatively, you can create a VM from scratch
  1. Download Ubuntu 16.04 Image
     1. X64: http://releases.ubuntu.com/16.04.6/ubuntu-16.04.6-desktop-amd64.iso
     2. X86: http://releases.ubuntu.com/16.04.6/ubuntu-16.04.6-desktop-i386.iso
  2. Create new VM

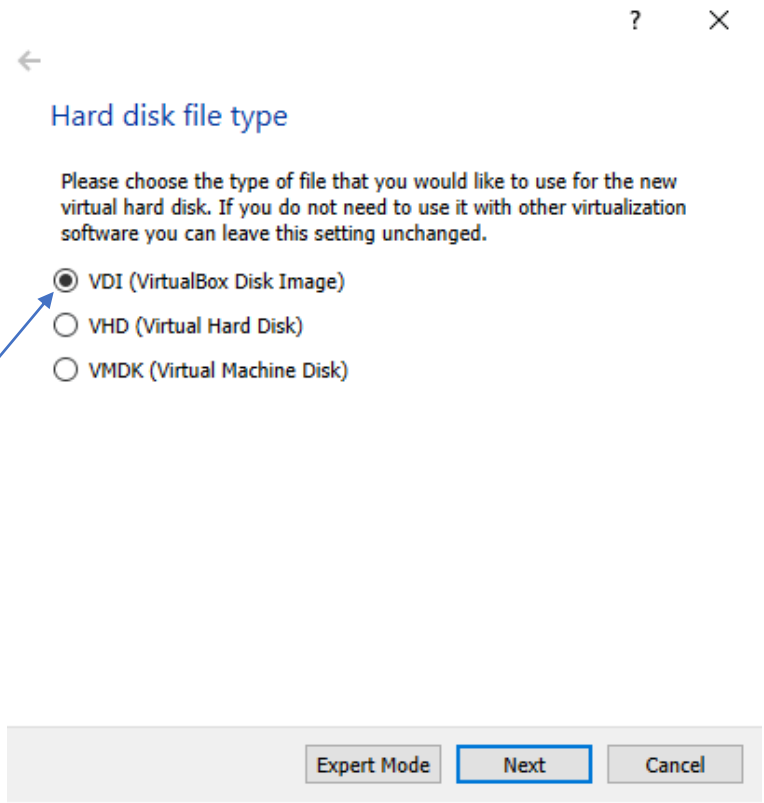
Virtual Machine From Scratch

1. Create new VM

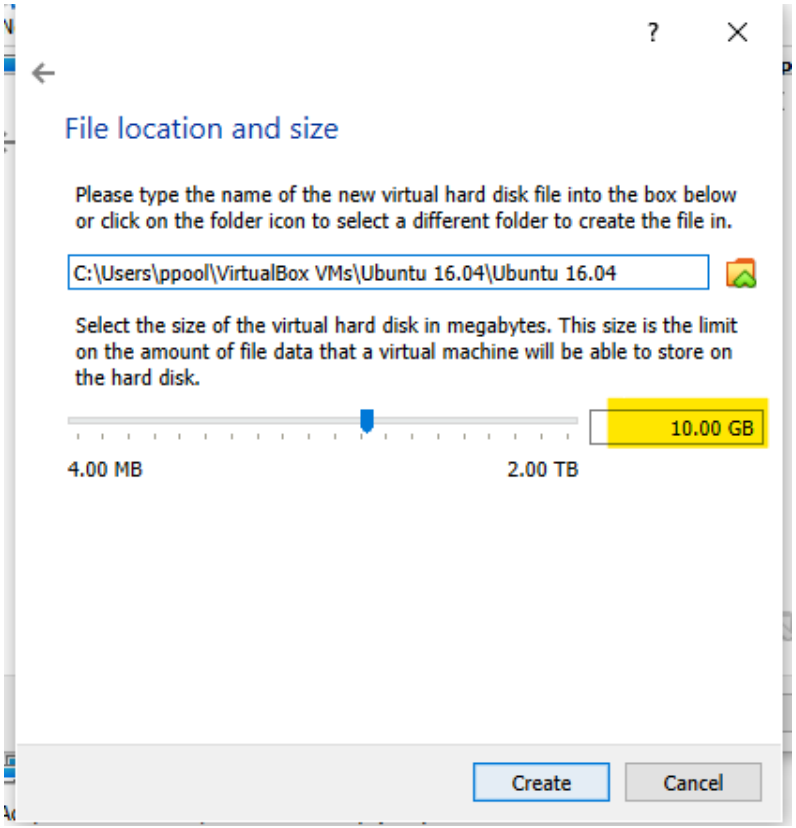


Create Disk


Hard disk
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the hard disk is 10.00 GB.
Do not add a virtual hard disk
Create a virtual hard disk now
Use an existing virtual hard disk file
MIE443_Turtlebot-disk001.vdi (Normal, 20.00 GB)
Create
Cancel

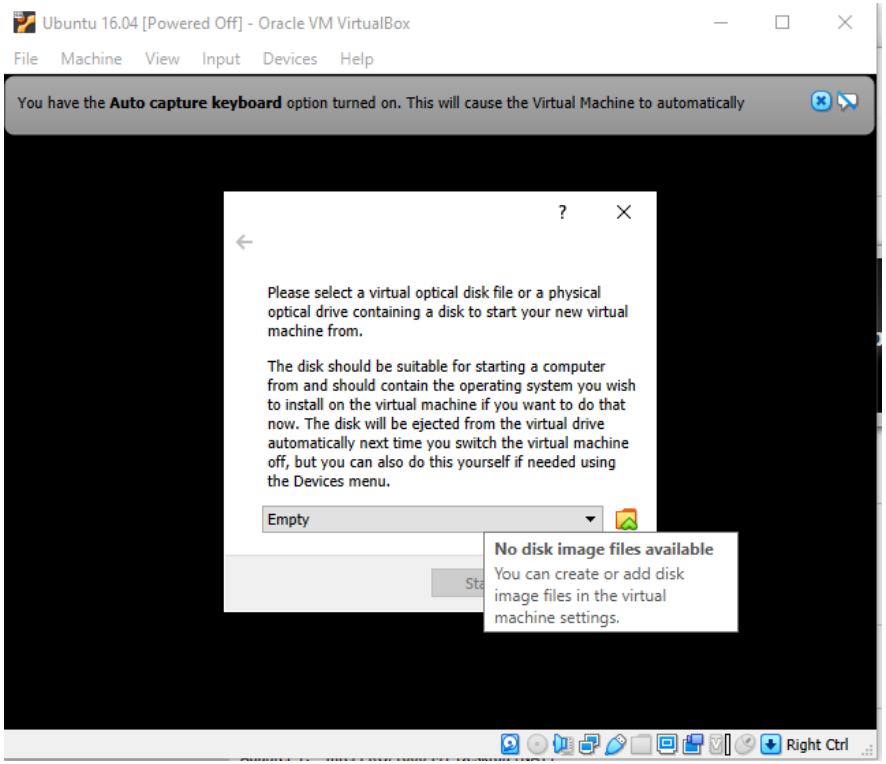

Hard disk file type
Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.
VDI (VirtualBox Disk Image)
VHD (Virtual Hard Disk)
VMDK (Virtual Machine Disk)
Expert Mode
Next
Cancel

Allocate Space



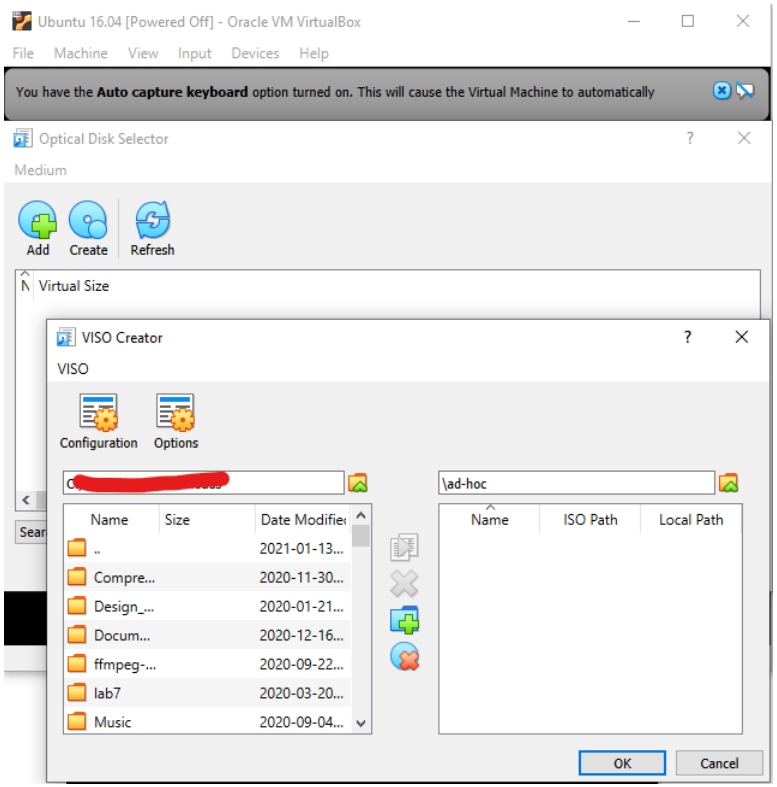
Install the OS

- It is an empty machine, you have to install an OS.



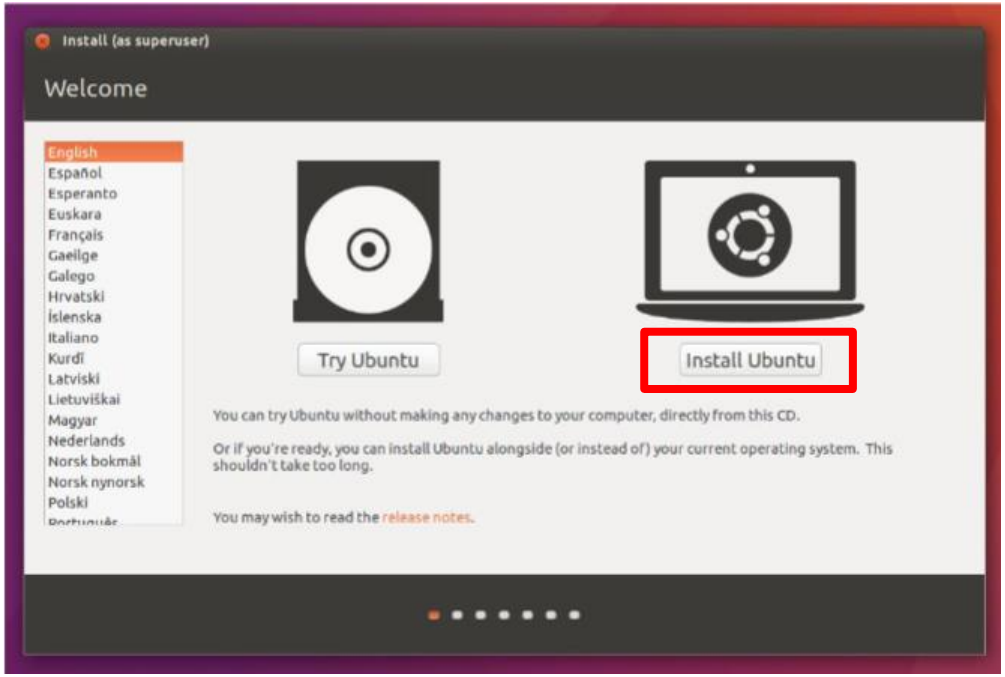
Import boot device

- Select and add the ISO you just downloaded as a boot device:



Install guest OS

- Install Ubuntu



Install ROS

- Install ROS & Turtlebot using the given script
  - Download the script and other given files
  - Open Terminal
  - > source /path/to/script/turtlebot_script.sh
  - If in Downloads: > source ~/Downloads/turtlebot_script.sh

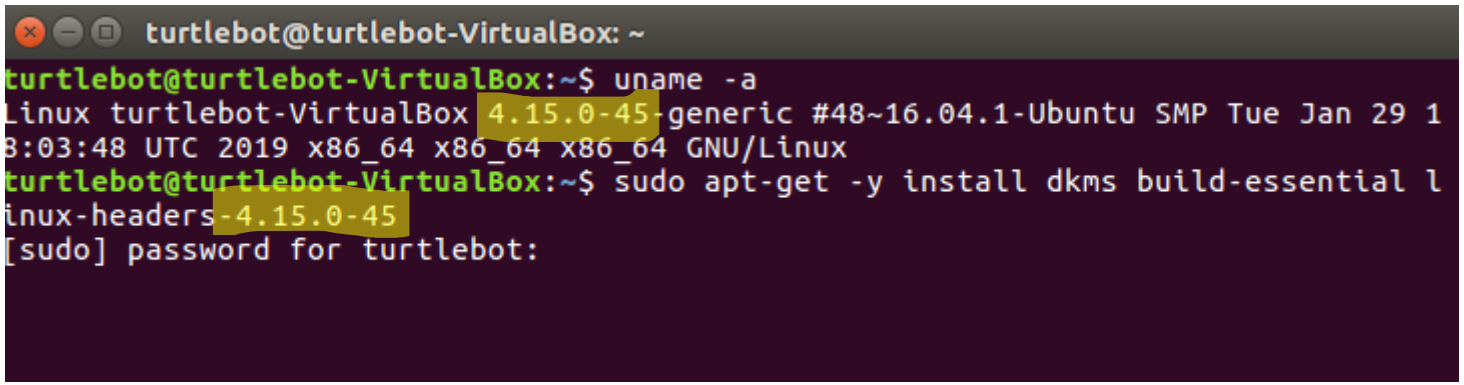
Install Guest Additions

- Optimizes the OS for better performance
  - Prepare linux by installing dependencies

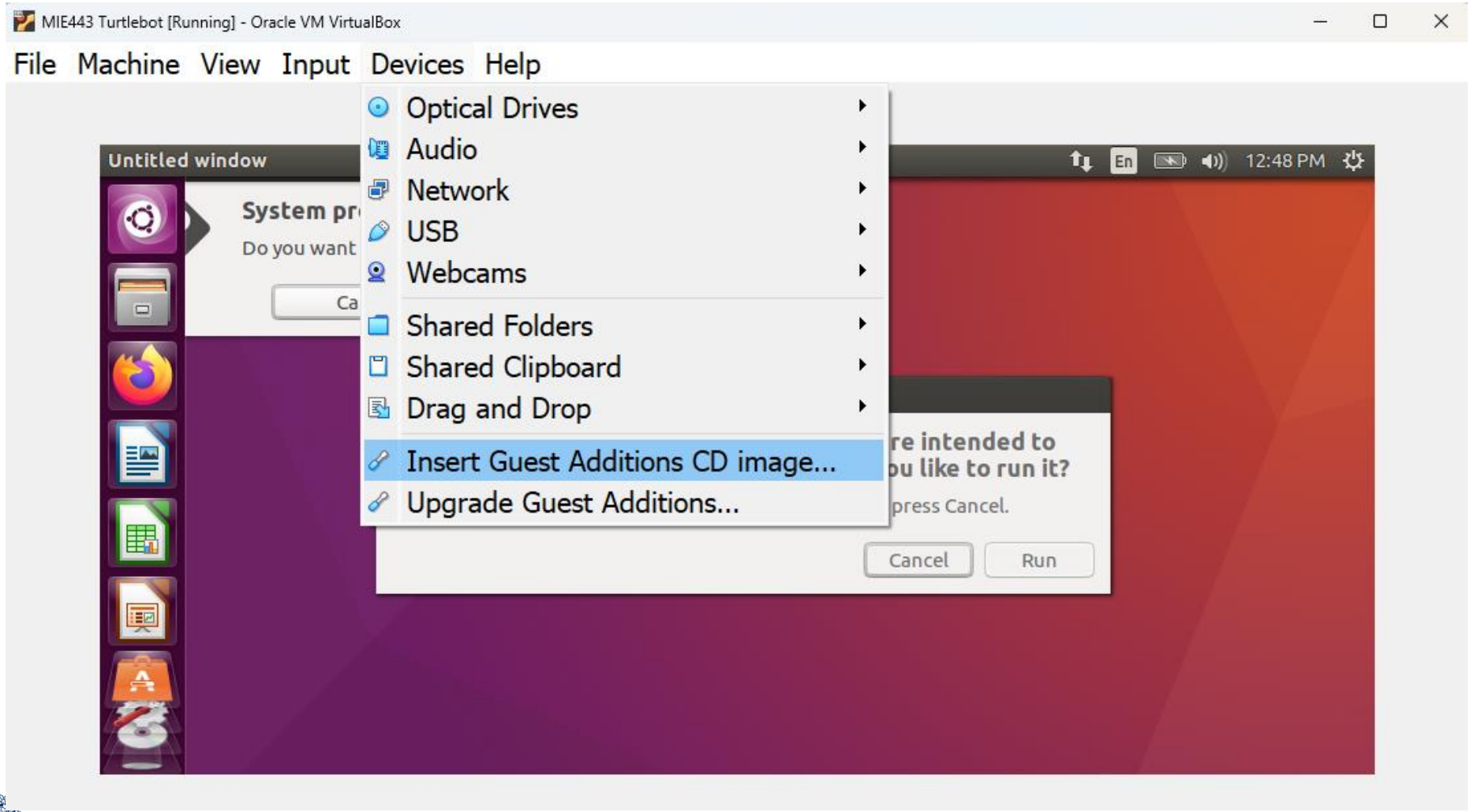
    $ uname –a #check version

    $ sudo apt-get -y install dkms build-essential linux-headers-VERSION #replace version with the one you get (the provided ova file's kernel version is 4.15.0-45

    $sudo reboot

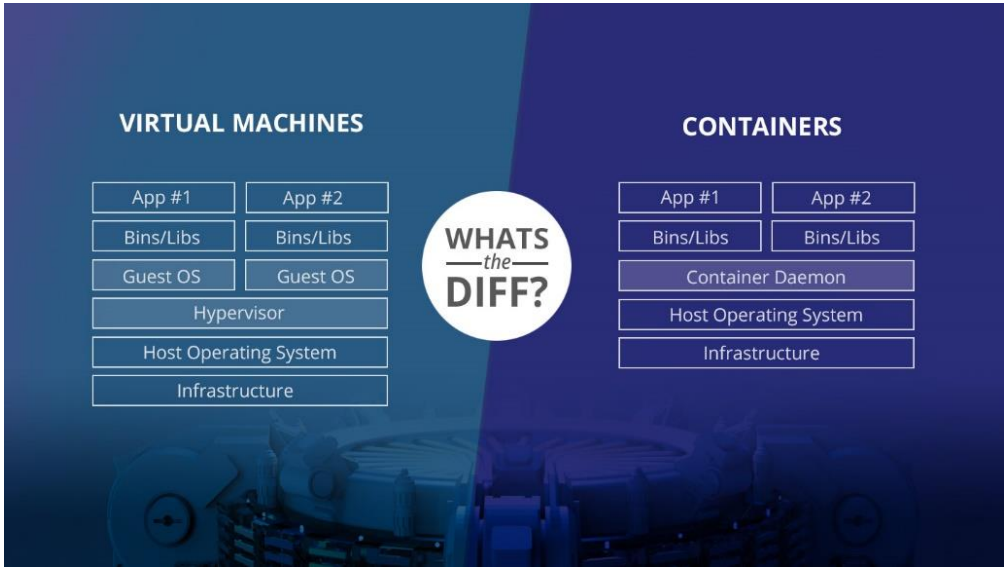


Install Guest Additions



Using Docker

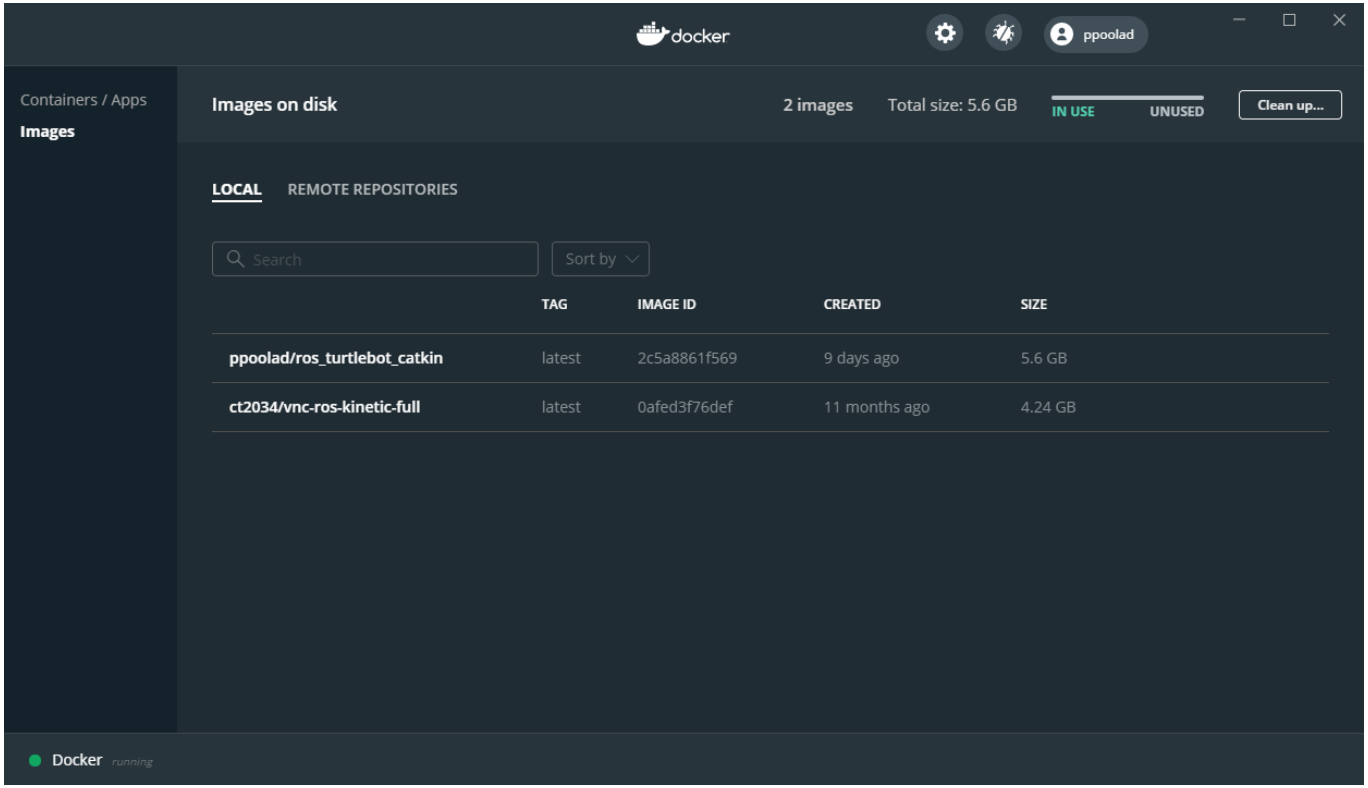
- Why?
- Do we need a whole VM?
  - Containers are a technology that provide us only what is necessary to run a certain application



Installing Docker

- Install Docker Desktop:
  - Get Started with Docker | Docker
- Restart PC if required
- Open a terminal (Powershell or WSL)
  - Pull the image we will be using:
    - > docker pull daaniiel/mie443-docker3
    - It takes couple of minutes to download and setup
  - Run the Image:
    - > docker run -v <your directory>:/mnt/ -it --rm -p 6080:80 -p 5900:5900 ct2034/vnc-ros-kinetic-full

Docker Desktop



Run a Container

```
ppoolad@SurfaceBook-Pooya:/mnt/c/Users/ppool$ docker run -v ~/MIE443:/mnt/MIE443 -it --rm -p 6080:80 -p 5900:590
0 ct2034/vnc-ros-kinetic-full
/usr/lib/python2.7/dist-packages/supervisor/options.py:297: UserWarning: Supervisord is running as root and it i
s searching for its configuration file in default locations (including its current working directory); you proba
bly want to specify a "-c" argument specifying an absolute path to a configuration file for improved security.
  'Supervisord is running as root and it is searching '
2021-01-14 03:44:08,967 CRIT Supervisor running as root (no user in config file)
2021-01-14 03:44:08,967 WARN Included extra file "/etc/supervisor/conf.d/supervisord.conf" during parsing
2021-01-14 03:44:08,984 INFO RPC interface 'supervisor' initialized
2021-01-14 03:44:08,984 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2021-01-14 03:44:08,984 INFO supervisord started with pid 23
2021-01-14 03:44:09,987 INFO spawned: 'xvfb' with pid 27
2021-01-14 03:44:09,988 INFO spawned: 'pcmanfm' with pid 28
2021-01-14 03:44:09,991 INFO spawned: 'lxpanel' with pid 29
2021-01-14 03:44:09,993 INFO spawned: 'lxsession' with pid 30
2021-01-14 03:44:09,995 INFO spawned: 'x11vnc' with pid 31
2021-01-14 03:44:09,999 INFO spawned: 'novnc' with pid 32
2021-01-14 03:44:11,029 INFO success: xvfb entered RUNNING state, process has stayed up for > than 1 seconds (st
artsecs)
2021-01-14 03:44:11,029 INFO success: pcmanfm entered RUNNING state, process has stayed up for > than 1 seconds
(startsecs)
2021-01-14 03:44:11,029 INFO success: lxpanel entered RUNNING state, process has stayed up for > than 1 seconds
(startsecs)
2021-01-14 03:44:11,029 INFO success: lxsession entered RUNNING state, process has stayed up for > than 1 second
s (startsecs)
2021-01-14 03:44:11,029 INFO success: x11vnc entered RUNNING state, process has stayed up for > than 1 seconds (
startsecs)
2021-01-14 03:44:11,029 INFO success: novnc entered RUNNING state, process has stayed up for > than 1 seconds (s
tartsecs)
```

Run a container

- Using VNCviewer on port 5900
- Opening browser on localhost:6080

Setup Catkin

- To have catkin folder set up run modified script “Turtlebot_script_catkinws_setup.sh”
  - Open Terminator
  - > source /path/to/script.sh

Notes

- Docker containers are not the same as VMs
  - **Docker container will be reset to the initial state after closing it**
    - If you modify something in the system, you need to save the state via docker commit
    - That is why mounting is important
  - Always save your file on host side (They will be deleted)!
  - VM is the suggested way for normal users.

VM References

- Oracle® VM VirtualBox®
- Docker Documentation | Docker Documentation
- Install WSL | Microsoft Learn

Ubuntu 16.04.6

The Underlying Operating System for the Robot Operating System (ROS)

UNIVERSITY OF TORONTO

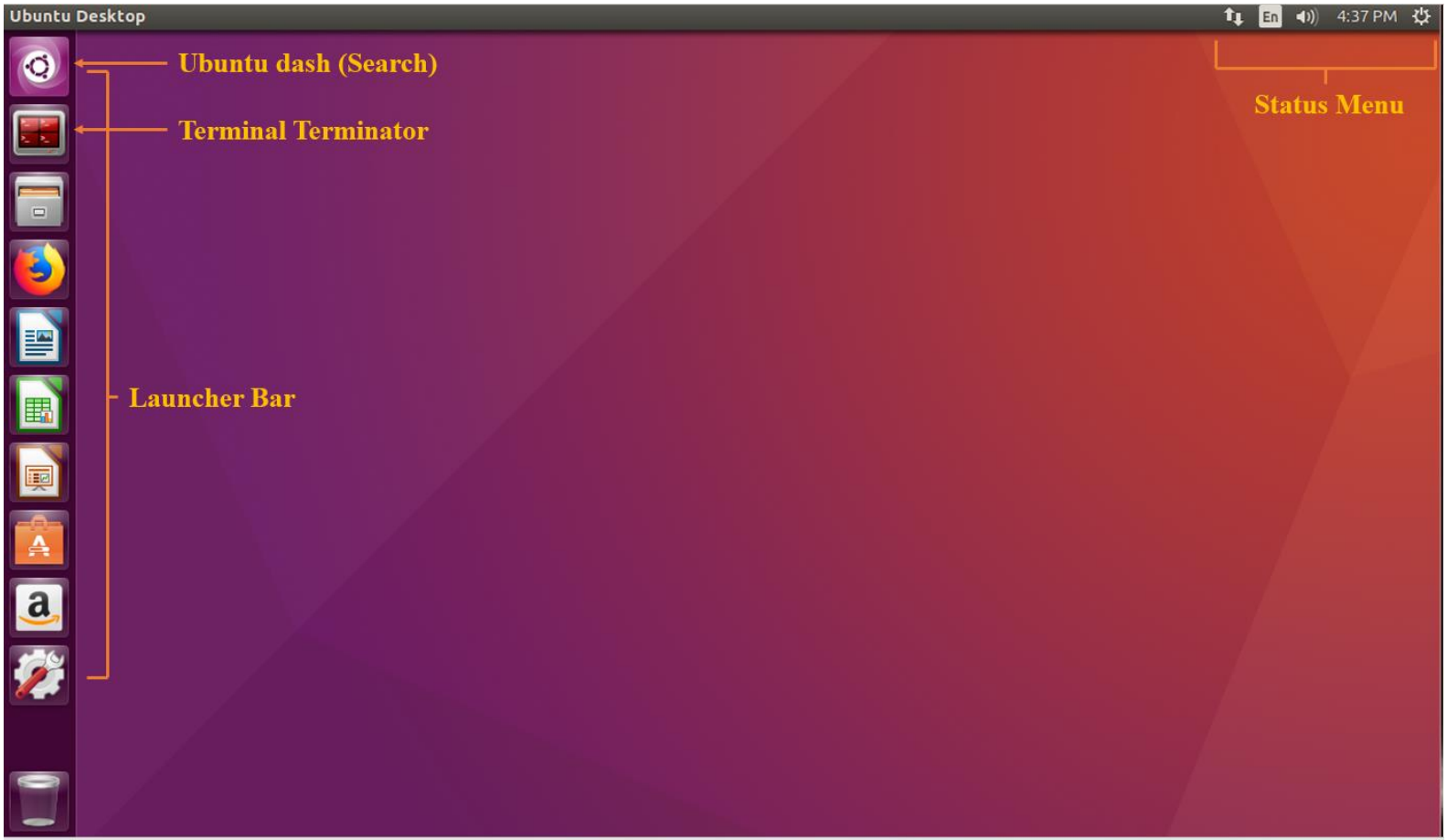
Ubuntu


- GNU/Linux Operating System Distribution.
- Widely used in the open source community for development.
- Supports a wide range of devices and drivers.
- Used by ROS as the base OS.

Operating System (OS)

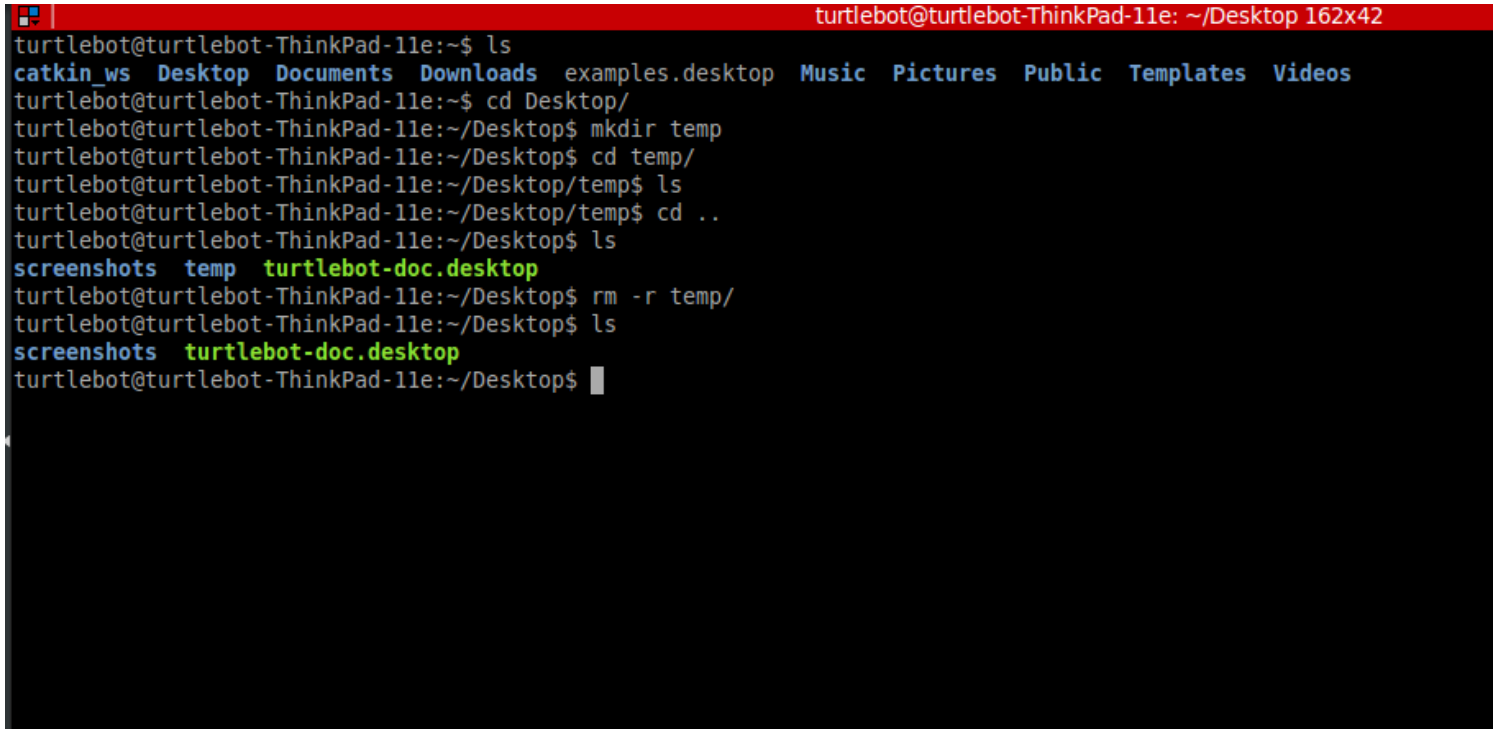
- Manages hardware and software resources.
- For **end users**: provides an **interface** to interact with applications.
- For **developers**: provides **hardware abstractions** (e.g., *file system*, *sockets*) and **services** (e.g., *printer spooler*, *logging daemon*) that eases software development.

GUI Shell (Unity)



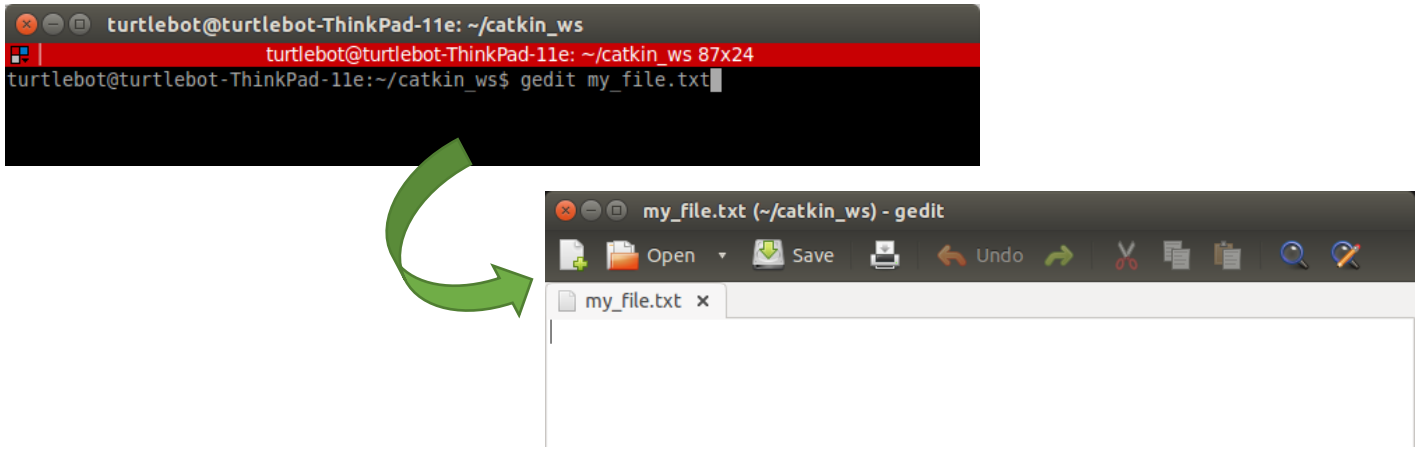
Command-Line Interface (CLI) Shell (Bash)

- Command-Line Interface that uses text commands to control the OS.
- More convenient for some activities.
- Syntax: `[program-name] [argument1] [argument2] … [argument n]`



Running a GUI program from terminal

- The terminal can also launch GUI based programs.
- The following example shows how to run Ubuntu text editor (**gedit**) from terminal:



- For this course, we recommend using either **gedit** (default editor) or **visual studio code**.

References: https://wiki.gnome.org/Apps/Gedit
https://code.visualstudio.com/

Some of the most used commands

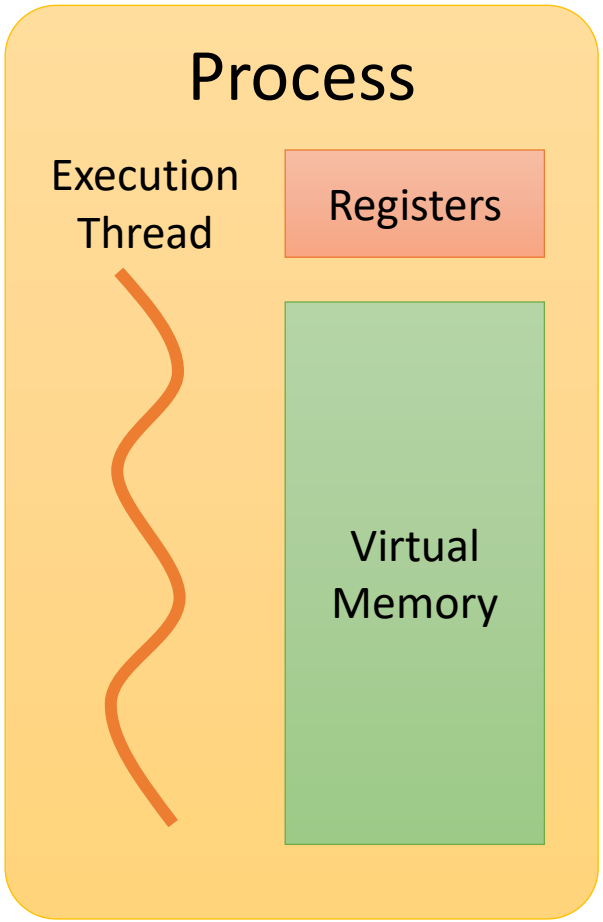
- `ls` – list all files within the current directory
- `cd` – change directory
- `mv` – move file to another directory
- `man` – shows the manual for a given program
- `mkdir` – make a directory.
- `rmdir` – remove a directory
- `touch` – creates an empty file
- `rm` – removes (deletes) a file
  - -r – remove recursively (multiple files)
  - -f – force remove (use with caution)
- `locate` – find a file within the filesystem
- `clear` – erases all text in the screen
- `sudo` – “super user do”, run a program with administrator privileges

CLI tips

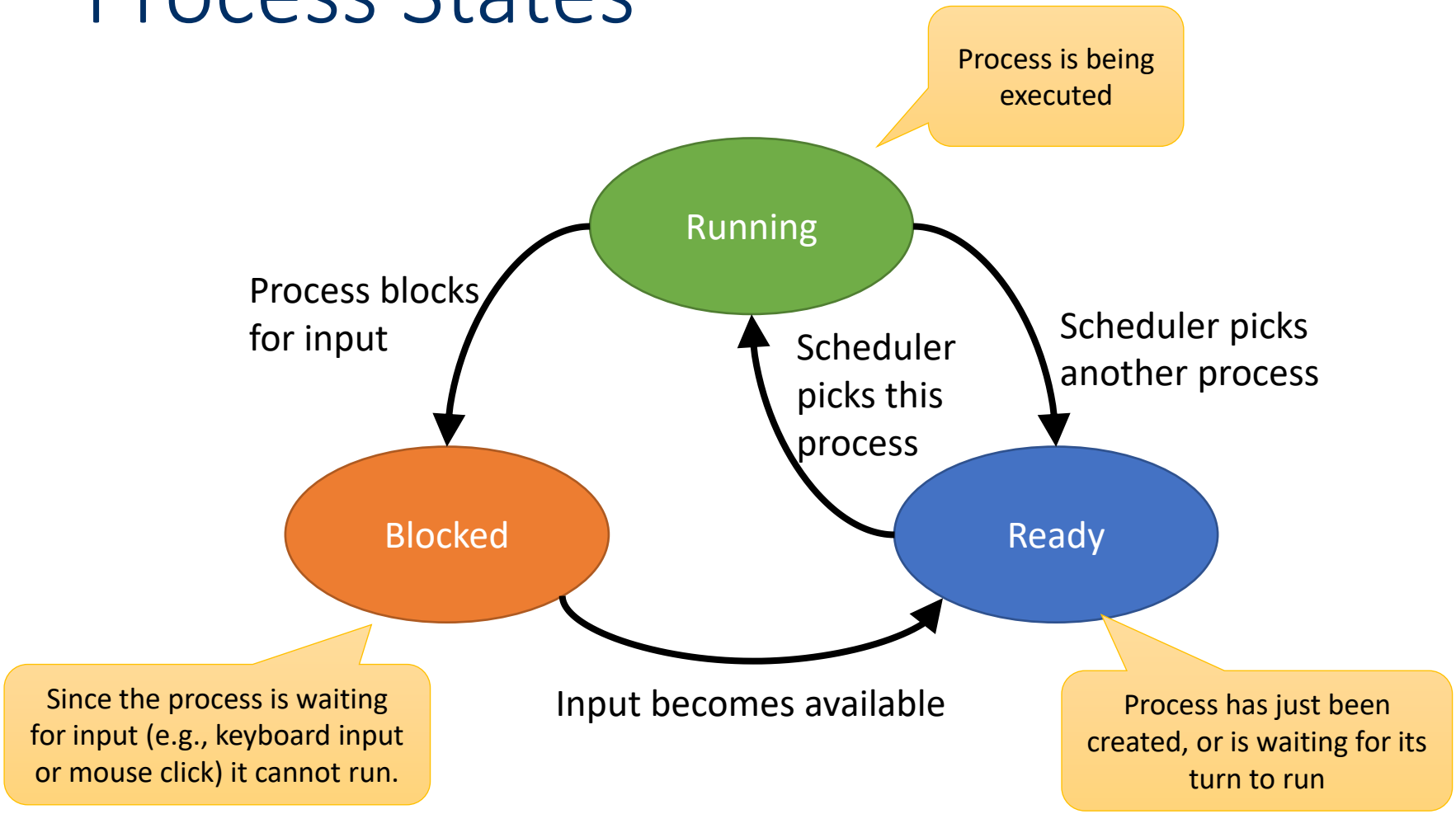
- **[Tab]** for completion.
- Double **[Tab]**to list programs or files.
- **[Ctrl]+[C]** to stop a program in the terminal.
- **[Ctrl]+[Insert]** and **[Shift]+[Insert]** to copy and paste on the terminal.
  - Right click will also copy/paste in the terminal
- You can use VIM to edit files directly in terminal
  - :q (exit w/o save)
  - :x (save and exit)
  - :<n> (go to line number)
  - ?<phrase> (search for phrase)
  - Press “i” to enter insert mode, “esc” to quit edit mode.

Processes

- When a program is started, a **process** is created.
- A process is the **instance of the program** that is being executed and contains the **program code and its activity**.
- A process can have one or multiple **threads that execute instructions**.
- **Processes do not share resources with each other.**

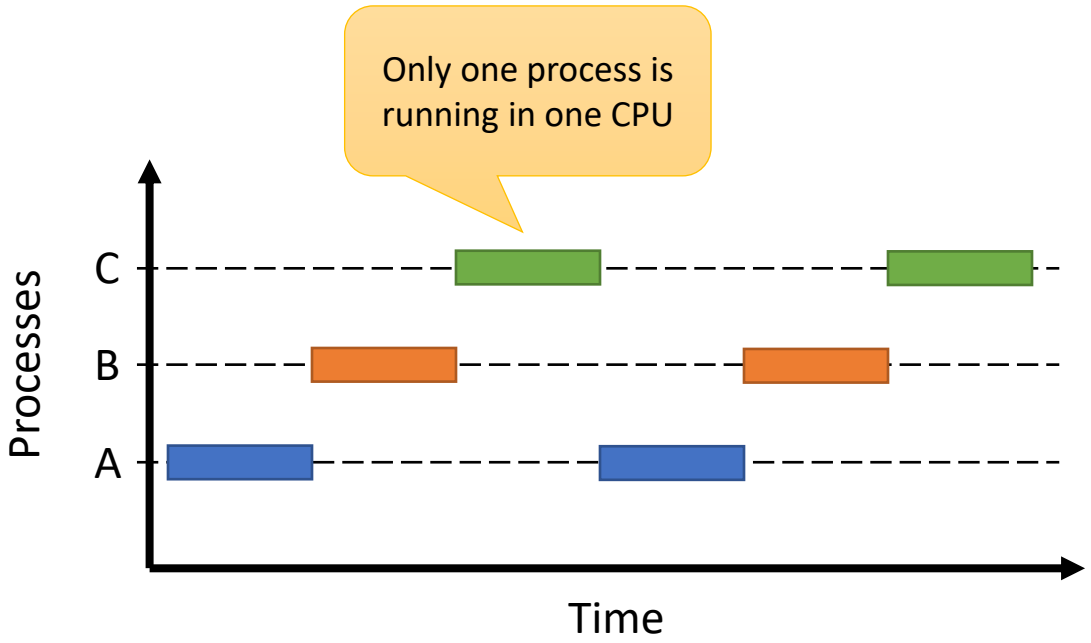


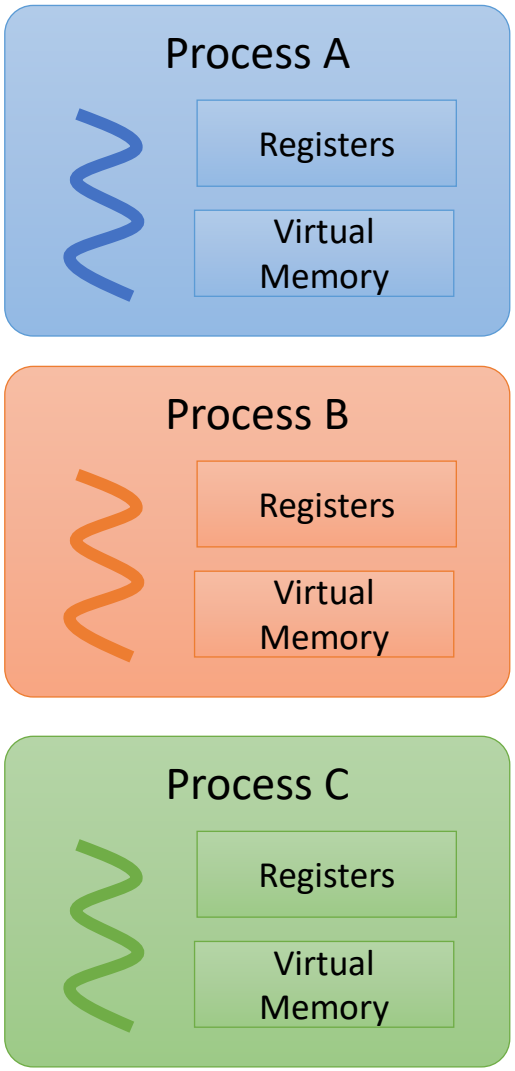
Process States



Adapted from: A. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. New Jersey: Pearson, 2014.

Process Model





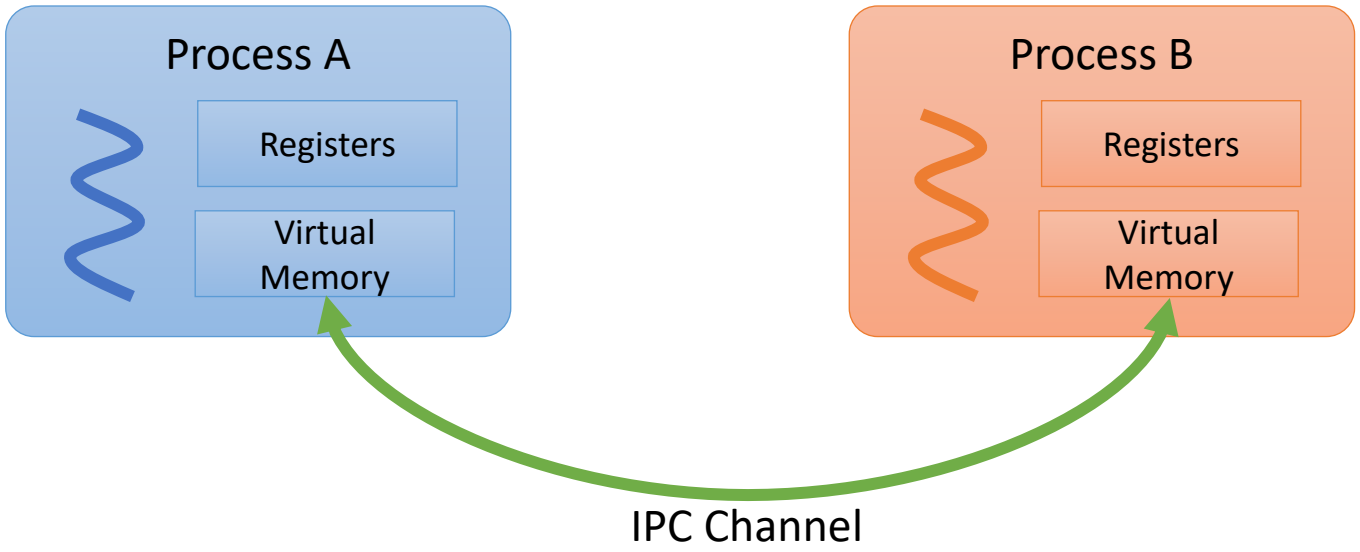
Adapted from: A. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. New Jersey: Pearson, 2014.

Resources are not shared between processes.

Inter-Process Communication (IPC)

- Inter-Process Communication (IPC) refers to mechanisms that **enable processes to share data**.
- The OS provides several IPC mechanisms (e.g., shared memory, sockets, pipes, message passing, etc.)
- In this course, we will rely on **ROS as the IPC mechanism**, which we will discuss later.

Inter-Process Communication (IPC)



Environment Variables

- Environment variables are **dynamic-named values that can affect how processes will behave**.
- On Unix-like systems (i.e., Ubuntu), each process has it own set of environment variables.
- A child process inherits a duplicate of the parent's environment variables.
- Example: the environment variable PATH stores the list of all directories with executable software, so that they can be found and executed from any directory.

The .bashrc file

- Bash (Ubuntu's CLI) runs the **initialization script** **`.bashrc`** every time it is opened by the User.
- It configures the shell to the User's preferences or needs.
- User-defined environment variables are set on the `.bashrc` file.
- Each user has their own `.bashrc` file on their home folder:

```
/home/user_name/.bashrc
```

Common commands in .bashrc

- Running configuration scripts with **source**:

```
source /path/to/script_file
```

- Setting up environment variables with **export**:

```
export <variable> = <value>

export <variable> = <value1> : <value2> : … : <valueN>

export <variable> = $<other_variable>

export <variable> = ${<other_variable>}
```

Example of a `.bashrc` file

.bashrc (~/) - gedit

Open Save

```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export TURTLEBOT_3D_SENSOR=kinect

```

Sets up ROS tools and the Catkin workspace

Sets up a variable named “TURTLEBOT_3D_SENSOR” with the value “kinect”, so that ROS know which sensor to use.

sh Tab Width: 4 Ln 125, Col 1 INS

UNIVERSITY OF TORONTO

References

- A. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. New Jersey: Pearson, 2014.
- https://help.ubuntu.com/16.04/ubuntu-help/index.html
- https://www.makeuseof.com/tag/ubuntu-an-absolute-beginners-guide/

The Robot Operating System

What is ROS?

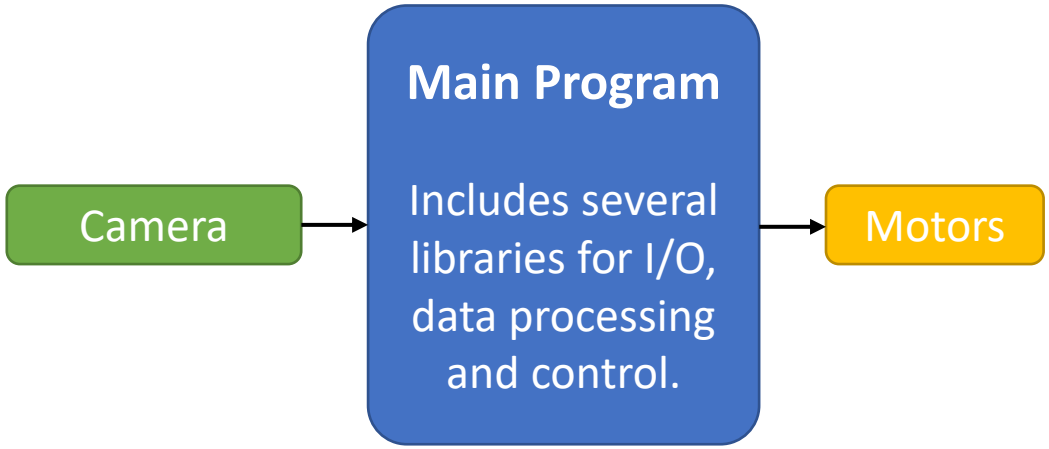
- The **Robot Operating System** is based on the Ubuntu GNU/LINUX distribution.
- ROS can be understood in two ways:
  - As a software, ROS is a **set of Open Source tools that enables different programs to communicate with each other**, helping users to create complex robot software architectures.
  - As a movement, ROS creates an **Open Source environment where researchers and companies can contribute robot-related algorithms**, which are released as packages.

Programming ROS

- Primarily using **C++** for development.
- Also supports **Python** for scripting.

Traditional development

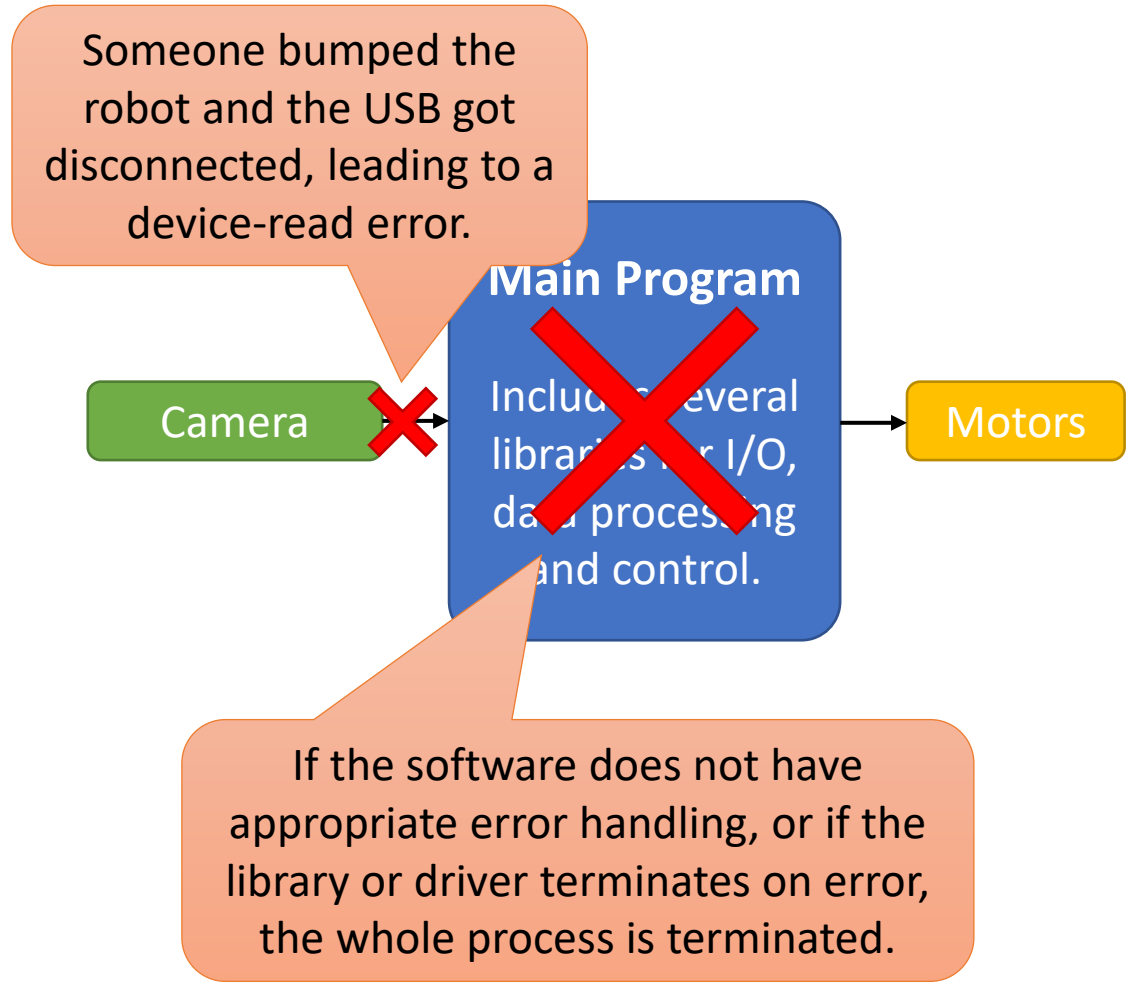
- A single program handles everything:
  - Read sensor data;
  - Process data;
  - Make decisions;
  - Control actuators;
- All software run on a single process.



Traditional development

Problems:

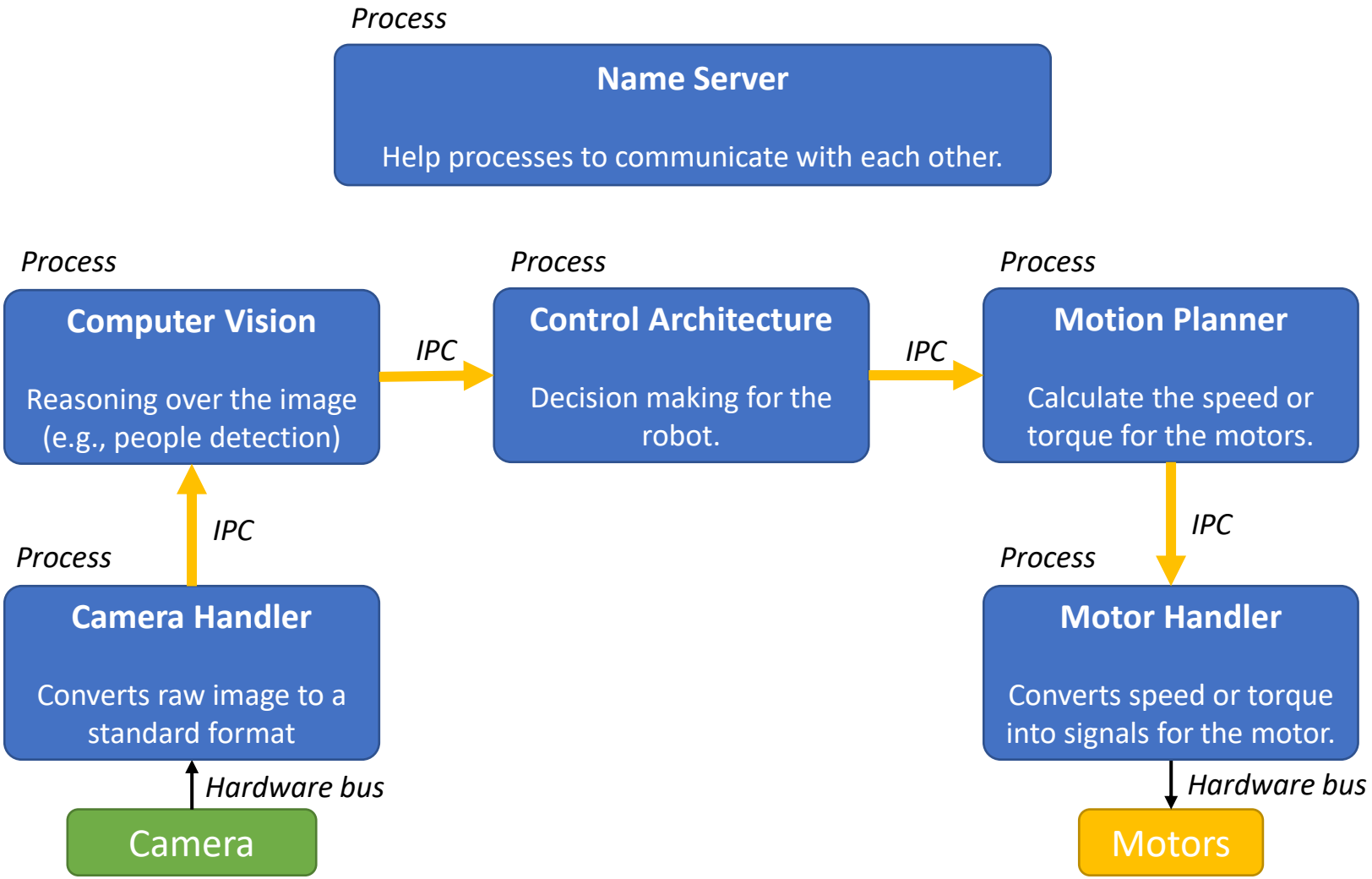
- The whole software has to be written in one language (e.g., c++).
- If there is **a bug** in a small part of the process that leads to failure, the **whole process will be terminated**.



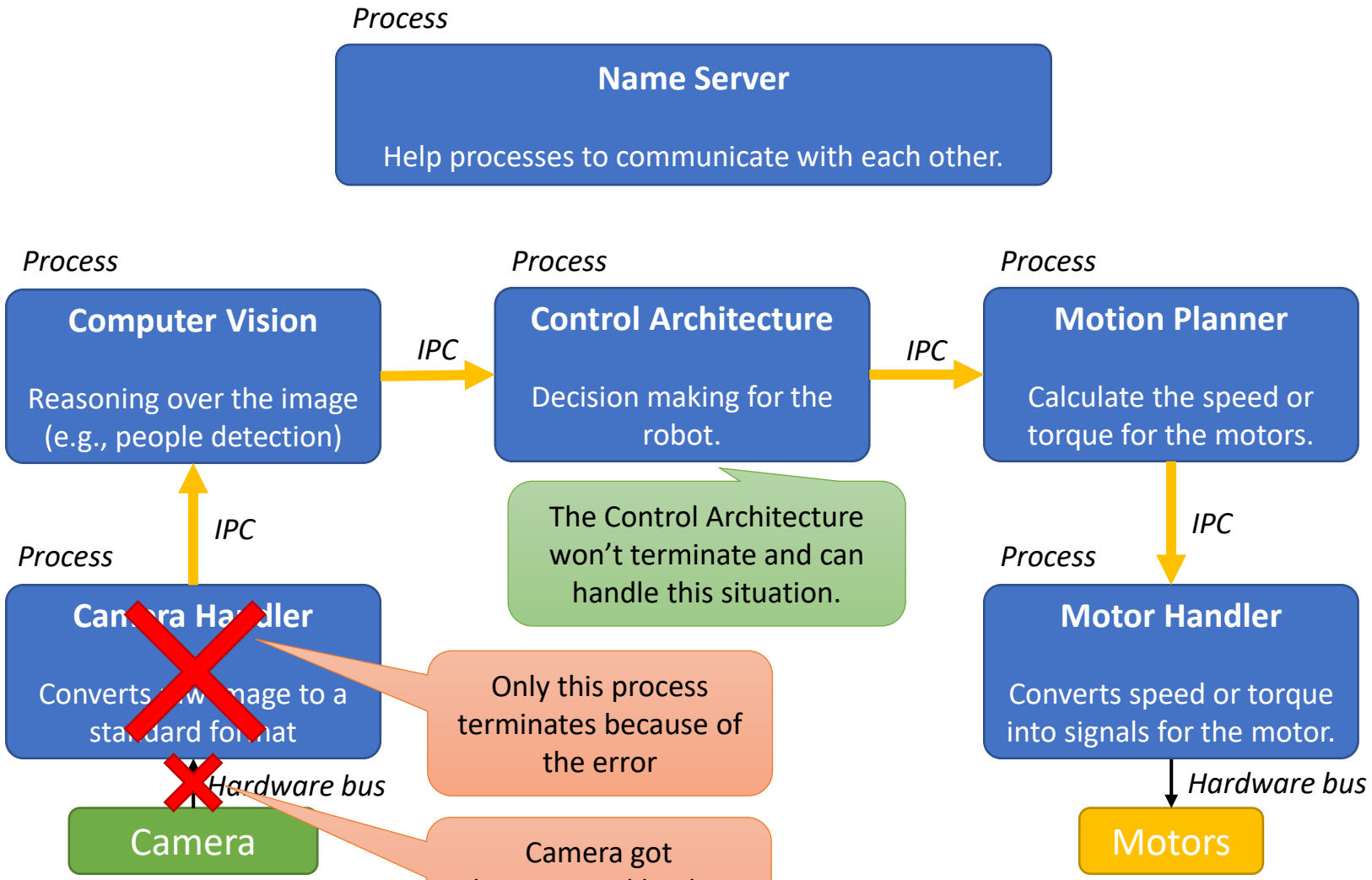
ROS-based Development

- Distributed software: **all different parts of the software run as separate processes:**
  - One process for per sensor.
  - One process per data processing algorithm.
  - One (or more) process for logic.
  - One process per actuator.
- Advantages:
  - Different languages can be used on each program.
  - If one process fail, the rest of the system is still working.

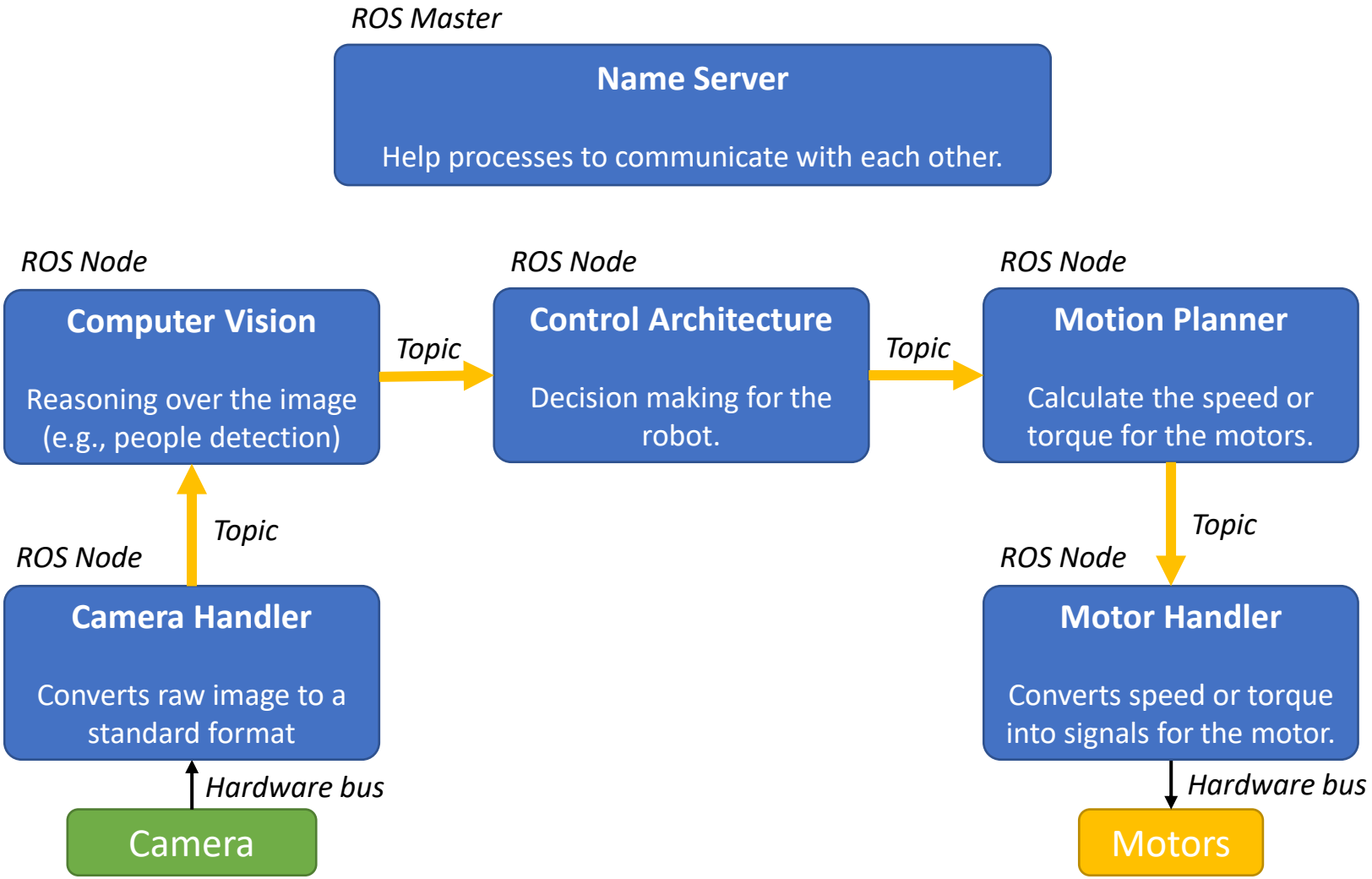
Distributed System Example



Distributed System Example During Failure



ROS System and ROS Notation



Main Concepts of ROS

- **ROS Master**

  Registers all **nodes** and provides them with the information they need.

- **Nodes**

  Executables that are run to perform actions and process data within the ROS framework. They communicate with one another using **topics**.

- **Topics**

  Named data buses over which nodes exchange **messages**.

- **Messages**

  Data structures.

ROS Master

- ROS provides a **nameservice system** which enables **Inter-Process Communication**. This system is called **ROS Master**.
- All ROS nodes have to connect to the ROS Master in order to make themselves known to other nodes.
- The ROS Master maintains a **list of all available topics and services**.
- Is brought up by starting `roscore` in terminal.

Reference: http://wiki.ros.org/Master

ROS Nodes

- All software running within the ROS framework are called nodes.
- Nodes can do any kind of work:
  - Read sensor data (e.g., camera node, bumper node).
  - Process sensor data (e.g., image-resize node, person-detection node).
  - Control hardware (e.g., motor node).
  - Make decisions (e.g., control architecture node).

Reference: http://wiki.ros.org/Nodes

ROS Nodes

- When a ROS Node connects to the ROS Master, it:
  - **Advertises** the topics and services it provides to other nodes, and informs their connection information.
  - Get the **subscription** (connection) information to the desired topics and/or services.

- With the subscription information, one node can connect to another node and **exchange messages directly** without passing through the ROS Master.

Reference: http://wiki.ros.org/Nodes

ROS Topics

- Topics are **one-directional data buses** in which data is exchanged.

- A node can be either a **Publisher** or a **Subscriber**.

- ROS topics and services are usually defined as:

  `/node_name/topic_name`

  `/topic_name`

Reference: http://wiki.ros.org/Topics

ROS Messages

- Messages are **strictly-typed data structures** that are communicated within different topics.
- Accessed by the topic callbacks within nodes.
- Examples:
  - `geometry_msgs/Twist`

    ```
    Vector3 linear
    Vector3 angular
    ```

  - `Vector3`

    ```
    Float64 x
    Float64 y
    Float64 z
    ```


Reference: http://wiki.ros.org/Messages

Default ROS Messages

- The default ROS Messages are the basis for all messages within ROS.
  - Commonly used messages:

    `String` stores text.

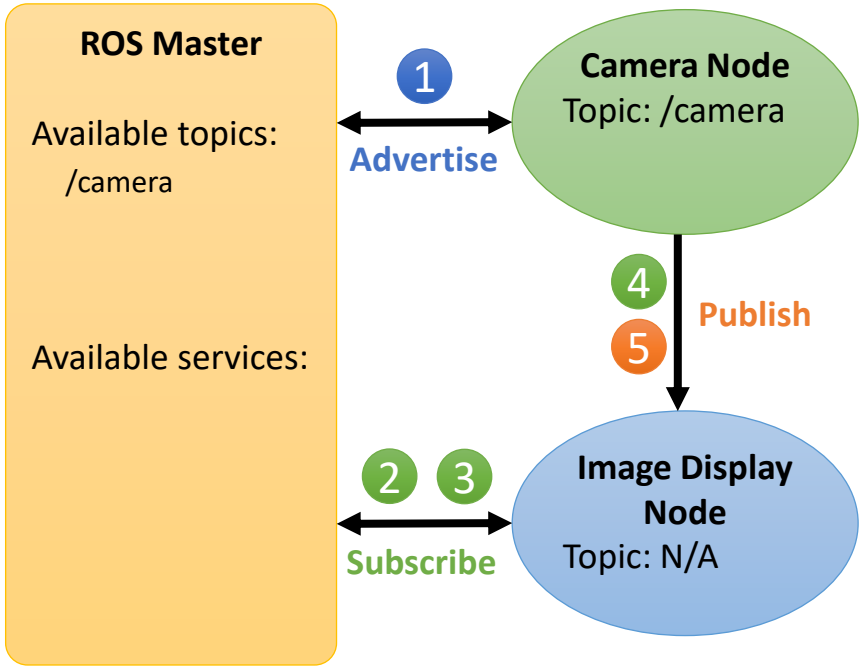
    `Header` stores the order and time information of a message. It is meant to be used by other message types.

    `Float32` stores a single floating point number.

    `Float32MultiArray` stores an array of `Float32` messages.
- All other messages are a combination of such data types.

Reference: http://wiki.ros.org/std_msgs

Example of Node Registration



Advertisement:

1. *Camera Node* contacts *ROS Master* and provides the connection information for its topic `/camera`

Subscription:

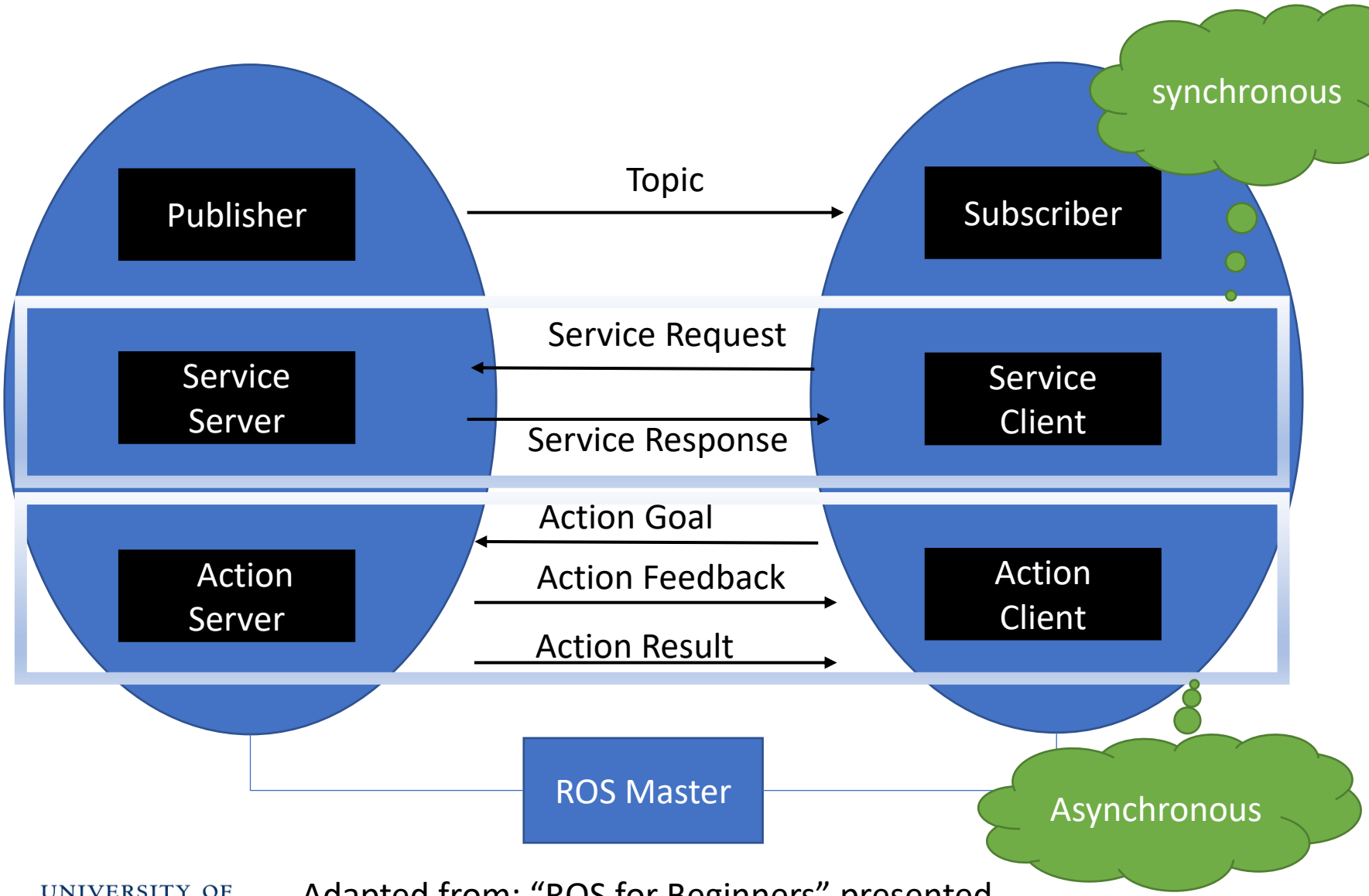
2. *Image Display Node* inquires *ROS Master* about `/camera`.
3. *ROS Master* provides the connection information.
4. *Image Display Node* connects to `/camera` topic

Publishing

5. *Camera Node* starts sending messages to the *Image Display Node* through the topic `/camera`

Reference: http://wiki.ros.org/Master

ROS Computation Graph



UNIVERSITY OF TORONTO

Adapted from: “ROS for Beginners” presented by Anis Koubaa

ROS Overview Summary

- ROS is a set of software tools that makes it easy to create distributed robot software.
- Each specialized software is called a **Node**.
- The **ROS Master** helps **Nodes** to find each other.
- **Nodes** communicate through **Topics** (or services).
- **Messages** are transferred within **Topics**.

Recent Versions of ROS

Version	Release Date	Poster
ROS Noetic Ninjemys	May 23rd, 2020	
ROS Melodic Morenia	May 23rd, 2018	
ROS Lunar Loggerhead	May 23rd, 2017	
ROS Kinetic Kame	May 23rd, 2016	
ROS Jade Turtle	May 23rd, 2015	
ROS Indigo Igloo	July 22nd, 2014	

Reference: http://wiki.ros.org/Distributions

How to Write Code for ROS

- When writing software for ROS, your source code must follow a structure that ROS can understand.
- To ease development, ROS uses a **building system called catkin**.
- catkin combines CMake macros and Python scripts to improve CMake's normal workflow.

Picture of a catkin. Courtesy of Wikipedia.

catkin Workspace

- A catkin workspace is where you put all your code.
- The catkin workspace is a directory (folder) that follows a directory structure that catkin can understand.
- It contains your code, the code catkin generates automatically, and the compiled code.

Creating a catkin Workspace

```
mkdir ~/catkin_ws/
```

Creates the root directory for your workspace. We will call it `catkin_ws`.

```
cd ~/catkin_ws/
```

Opens the `catkin_ws` directory.

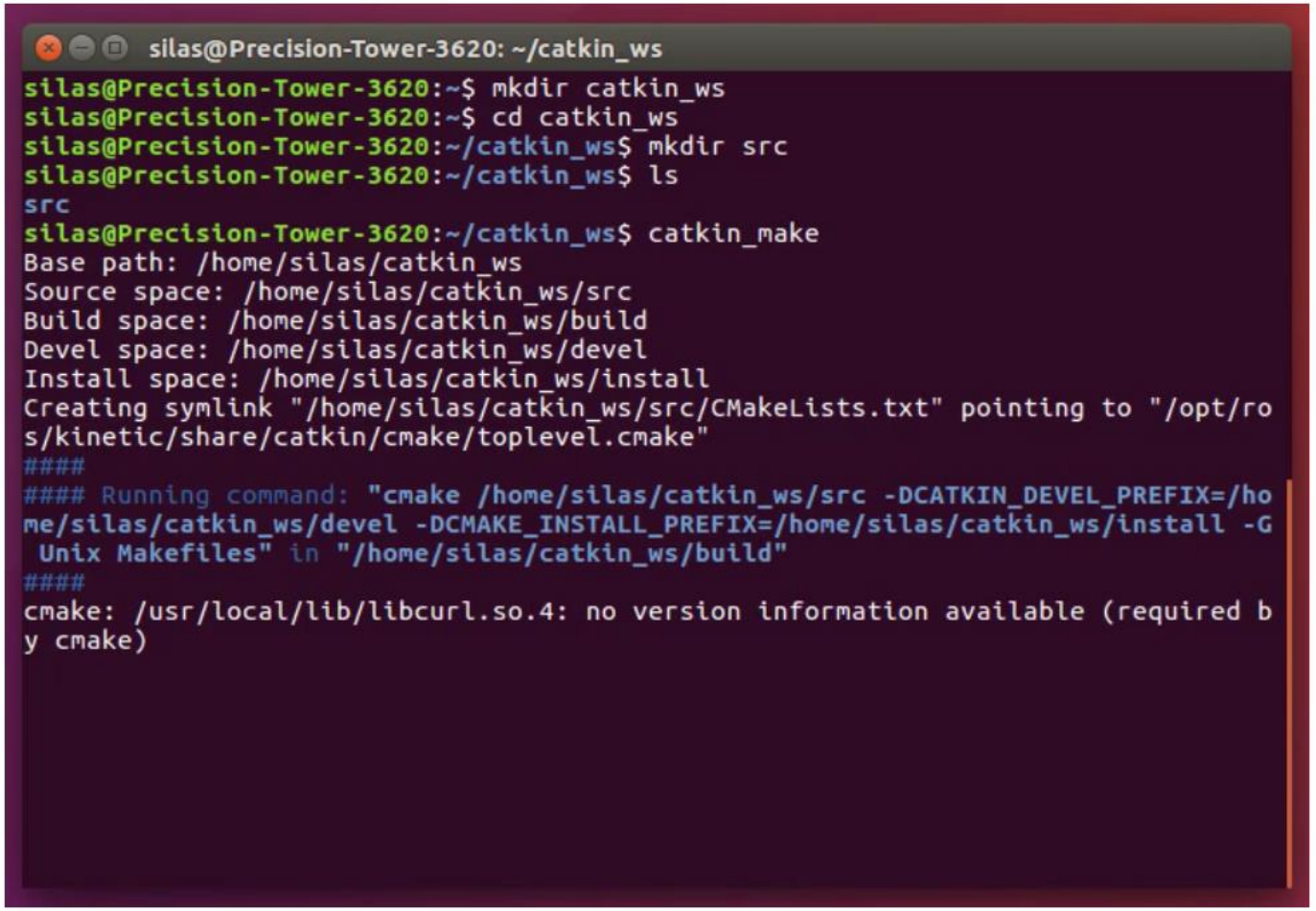
```
mkdir src
```

Creates a directory called `src` inside `catkin_ws`.

```
catkin_make
```

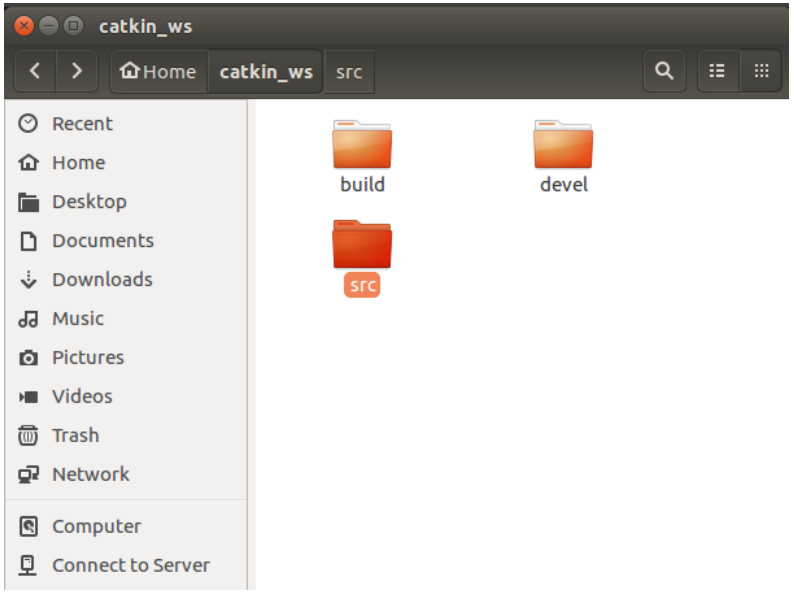
Invokes the catkin build system. Since there is only an empty `src` directory and nothing else, it will configure the workspace.

Creating a catkin workspace



catkin Workspace

- There are three sub-directories:
  - `src` contains the source code of the user's packages.
  - `devel` contains the automatically generated code, setup scripts, and information about the user's packages.
  - `build` where the output binaries (executable) are stored.



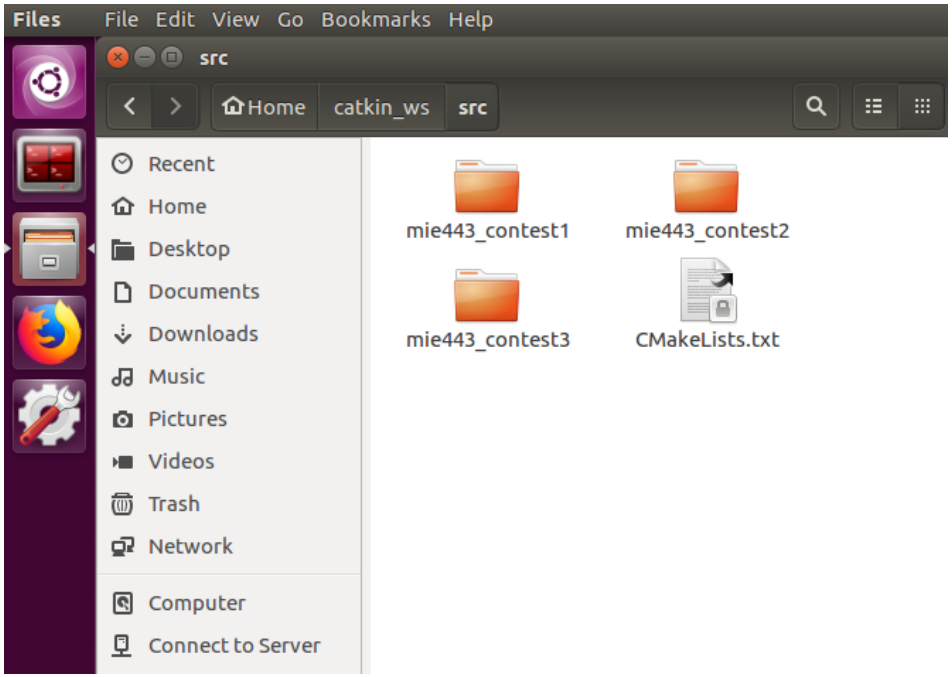
Reference:
http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

Does my code go into ‘src’?

- Yes, but it should be organized as packages.
- We use “catkin package”, “ROS package”, and “package” interchangeably.
- A package is a directory which follows a structure catkin can understand.
  - `src` directory for source code.
  - `include` directory for the header files (source code).
  - `package.xml` with information about your code for catkin.
  - `CMakelists.txt` with information about your code for CMake.

Example of Packages

- Your src folder can contain one or more packages, as in the example below:



How to create a package

- ROS uses Catkin to automatically generate the source code required to enable IPC, therefore we need to use its tools to create our package.

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3] …
```

- Example:

```
$ catkin_create_pkg myPackage std_msgs rospy roscpp
```

A directory named **my_package** will be created under **…/src/** with all the automatically generated directories and files.

Allows the project to use the basic standard messages provided by ROS

Enables Python 2.7 scripting

Enables the C++ compiler

Reference: http://wiki.ros.org/ROS/Tutorials/CreatingPackage

Creating a catkin package

```
$ cd ~/catkin_ws/src
```

Opens the `src` directory.

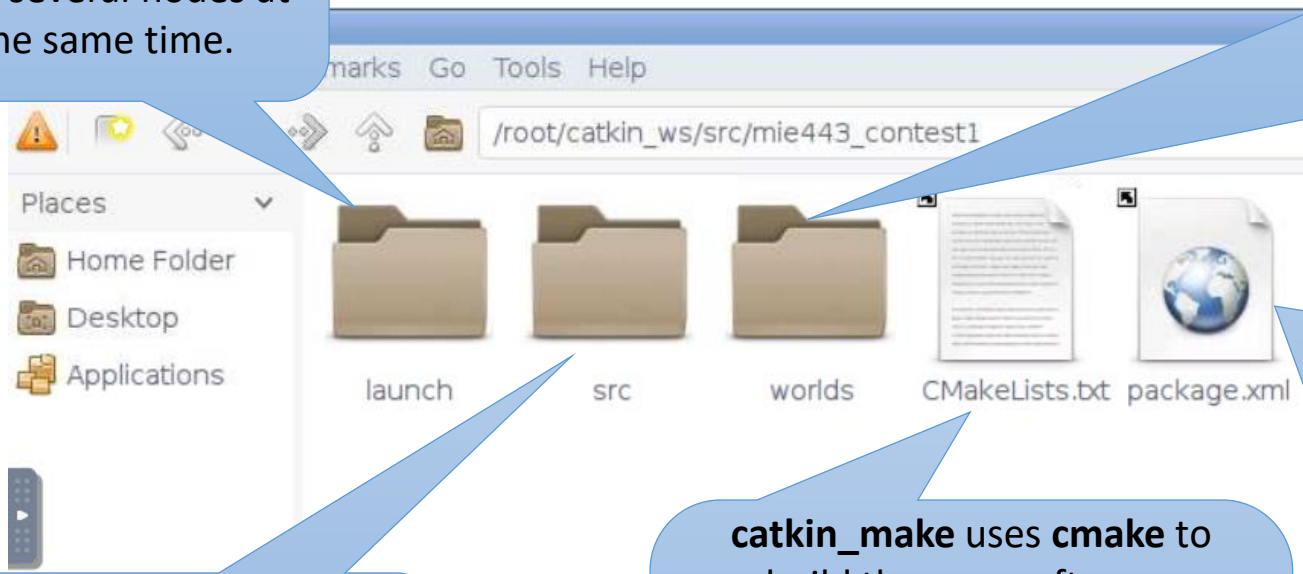
```
$ catkin_create_pkg my_package std_msgs
rospy roscpp
```

Invokes the catkin system to create a package named “my_package”, which depends on `std_msgs`, `rospy` and `roscpp`

What does a package look like?

Contain all the launch files (XML) used to start several nodes at the same time.

Particular for this project. Contains the map of the Contest 1 scene used in the Gazebo simulator (refer to the *Student Manual* on Quercus).



Contains information regarding the package (package name, authors' names and e-mails, etc.) and the required libraries. Used by **Catkin**.

Where the source code (*.cpp) of the nodes is located.

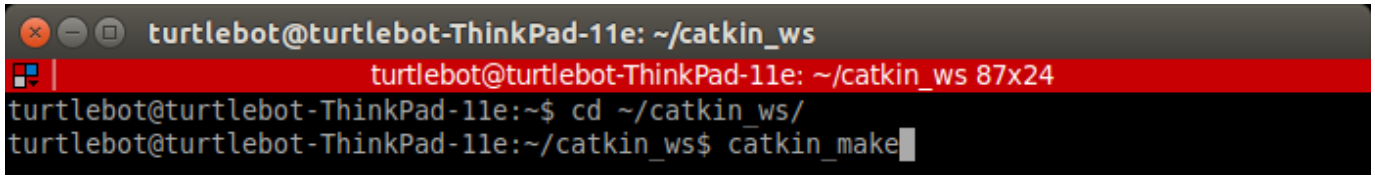
catkin_make uses **cmake** to build the user software. Therefore, the user is required to provide the CMakeLists.txt file which contains the information (names, directories, etc.) of all the libraries that are used by the package.

UNIVERSITY OF TORONTO

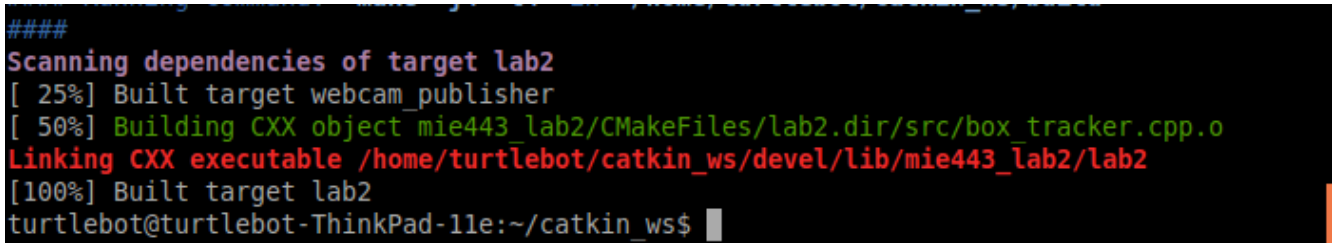
Building the workspace

- `Catkin` is the ROS build system.
- `catkin_make` compiles the workspace. It is run from the terminal while within the workspace directory in terminal.

```
$ cd ~/catkin_ws
$ catkin_make
```



- When it is properly compiled you will see the following lines in the terminal when the command finishes.



What does the code of a ROS node look like? (1)

1. Initiate the ROS node through:
   ```
   ros::init(argc, argv, nodeName)
   ```

2. Create one node handler using:
   ```
   ros::NodeHandle()
   ```

3. Use node handler to advertise topics and services using:
   ```
   ros::NodeHandle.advertise<type>(topicName, queueSize)
   ```


Reference:
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

What does the code of a ROS node look like? (2)

4. Use node handler to subscribe to topics or services with:

```
ros::NodeHandle.subscribe(topicName, queueSize, callbackFunc)
```

5. Call `ros::spin()` or `ros::spinOnce()` to enable the node to receive data from other nodes.

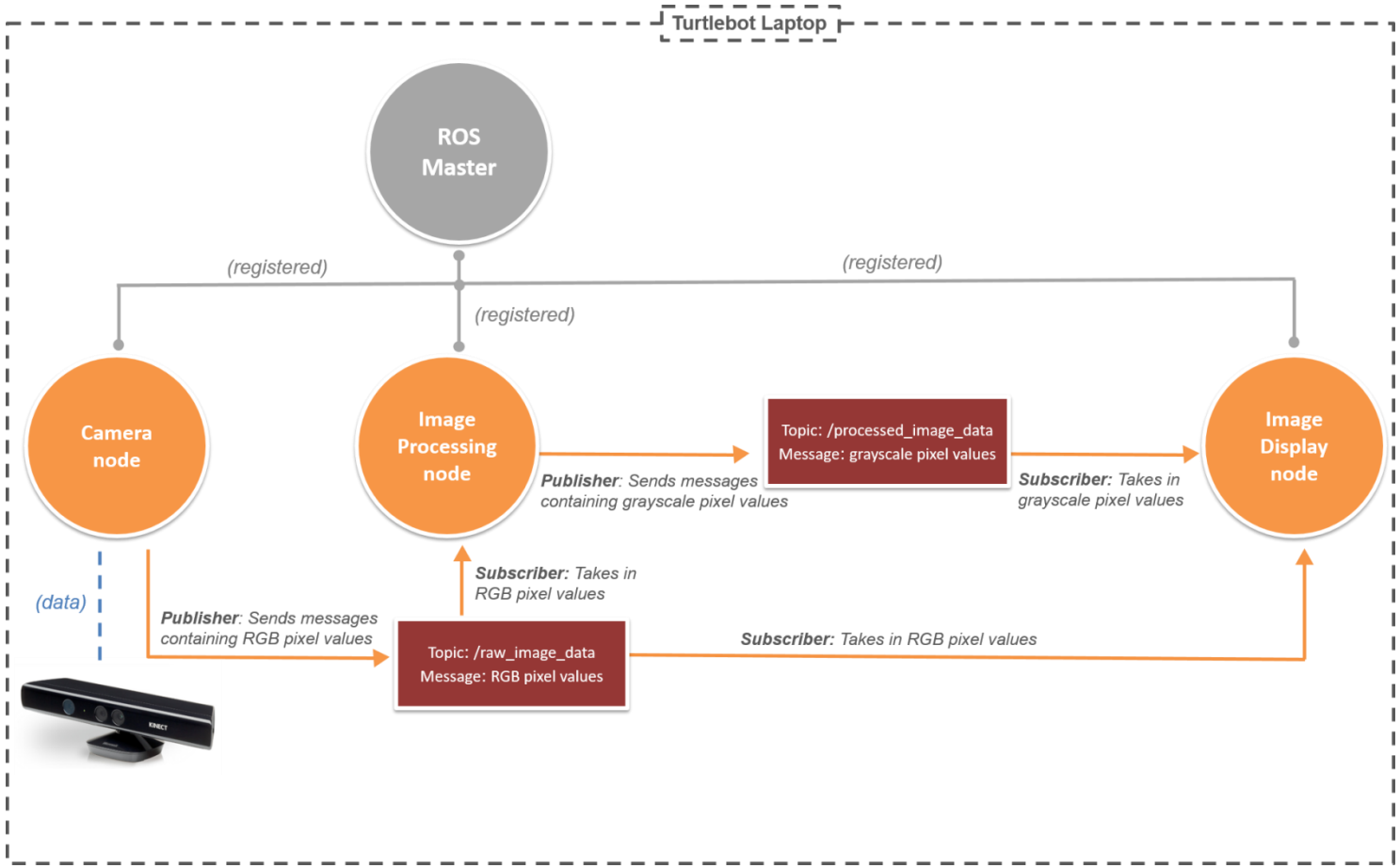
ros::spin() and ros::spinOnce()

- `ros::spin()` is an eternal loop, and is used when you do not need to process any data other than the messages your node receives.

- `ros::spinOnce()` checks for new messages only once, allowing your code to process other data. Needs to be called repeatedly, otherwise messages will be lost.

Reference:
http://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning

Example of Robot Software Architecture using ROS



GAZEBO

- A 3D dynamic simulator to simulate robots in indoor and outdoor environments.

Uses of Gazebo:

- Testing robotics algorithms,
- Designing robots

Features of Gazebo:

- Multiple physics engines,
- A rich library of robot models and environments,
- A wide variety of sensors,
- Convenient programmatic and graphical interfaces

Reference: http://gazebosim.org

Using Gazebo to Implement the Contest Code

- roslaunch mie443_contest1 turtlebot_world.launch world:=practice

Launching the Simulated World

- roslaunch mie443_contest1 gmapping.launch

ROS Gmapping Module

- rosrun mie443_contest1 contest1

Run the Code in Simulation

- rosrun map_server map_saver -f /home/turtlebot/

Saving the Map

Contest 1 source code (1)

```
#include <ros/console.h>
#include "ros/ros.h“

#include <geometry_msgs/Twist.h>
#include <kobuki_msgs/BumperEvent.h>

#include <sensor_msgs/LaserScan.h>

#include <stdio.h>
#include <cmath>
#include <chrono>
```

Includes the ROS libraries

Includes the message used by the publisher that will be created.

Includes the messages used by the subscribers.

Adds the Standard I/O library for printing and reading text from the console, the mathematic functions and a library for reading the computer time.

Contest 1 source code (2)

A callback function is used to subscribe to a single topic. This topic subscribes to the bumper.

The parameter of the callback must be the message published by the desired topic.

```
void bumperCallback(const kobuki_msgs::BumperEvent::ConstPtr& msg){
        //fill with your code
}


void laserCallback(const sensor_msgs::LaserScan::ConstPtr& msg){
        //fill with your code
}
```

When a new message is received, the callback function is called. The message is sent as a parameter.

Contest 1 source code (3)

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "image_listener");
  ros::NodeHandle nh;

  ros::Subscriber bumper
    "mobile_base/events/bumper", 10, &bumperCallback);
  ros::Subscriber laser_sub = nh.subscribe(
    "scan", 10, &laserCallback);
  ros::Publisher vel_pub = nh.advertise<geometry_msgs::Twist>(
    "cmd_vel_mux/input/teleop", 1);

  ros::Rate loop_rate(10);
```

Starts the ROS Node with the name **image_listener** and creates a new Node Handler to publish and subscribe to topics.

Subscribes to the bumper topic of the Turtlebot. To that end, the name of the topic, the message queue size, and the address of the callback function should be provided.

Advertises a new topic named **cmd_vel_mux/input/teleop** with a queue with size 1 that publishes messages of the type **geometry_msgs/Twist**.

Help us to create a loop that outputs messages at a 10 Hz rate.

Contest 1 source code (4)

```
geometry_msgs::Twist vel;

// contest count down timer
std::chrono::time_point<std::chrono::system_clock> start;
start = std::chrono::system_clock::now();
uint64_t secondsElapsed = 0;

float angular = 0.0;
float linear = 0.0;
```

The message **vel** that will be published.

Gets the current time of the system. Used later to monitor how much time the software has been running.

The variables that hold the **angular** and **linear** velocities of the robot.

Contest 1 source code (5)

```
    while(ros::ok() && secondsElapsed <= 900){
        ros::spinOnce();
        //fill with your code


        vel.angular.z = angular;
        vel.linear.x = linear;
        vel_pub.publish(vel);


        // The last thing to do is to update the timer.
        secondsElapsed = std::chrono::duration_cast<std::chrono::seconds>(
           std::chrono::system_clock::now()-start).count();
        loop_rate.sleep();
    }

    return 0;
}
```

Checks if the user has hit [CTRL]+[C] on the keyboard and if the node is running for less than 15 minutes.

Uses **ros::spinOnce()** to receive all messages from the publishers and calls the callback functions if necessary.

Updates the message **vel** with the values of **angular** and **linear**, and then publishes them.

Updates the timer.

Wait enough time so that the loop is executed at 10 Hz.

Building the workspace (revisited)

- Run **`catkin_make`** in the workspace directory.

- **Automatic code generation**
  - If there is a **`msg`** directory and it is referenced on the `CMakeLists.txt` file, `catkin_make` will read all the user defined messages and automatically generate the C++ and Python code for them.
  - Same for **`srv`** directory, which contains the services.
  - Topics do not need to be described separately.

Reference: http://wiki.ros.org/catkin/commands/catkin_make

Building the workspace (revisited)

- **Compilation:** `catkin` will use `cmake` and custom Python scripts to generate the build configuration and compile the whole project.

- If new messages or services are created, you need to tell ROS by running:

```
$ source ~/catkin_ws/devel/setup.bash
```

Running the code

- Open the terminal.
- Run `roscore` to initiate the ROS Master.
- Use `rosrun` or `roslaunch` to initiate all supporting nodes:
  - Turtlebot node
  - Microsoft Kinect node
  - Simulator node
- Use `rosrun` or `roslaunch` to initiate your node.

ROS command-line tools

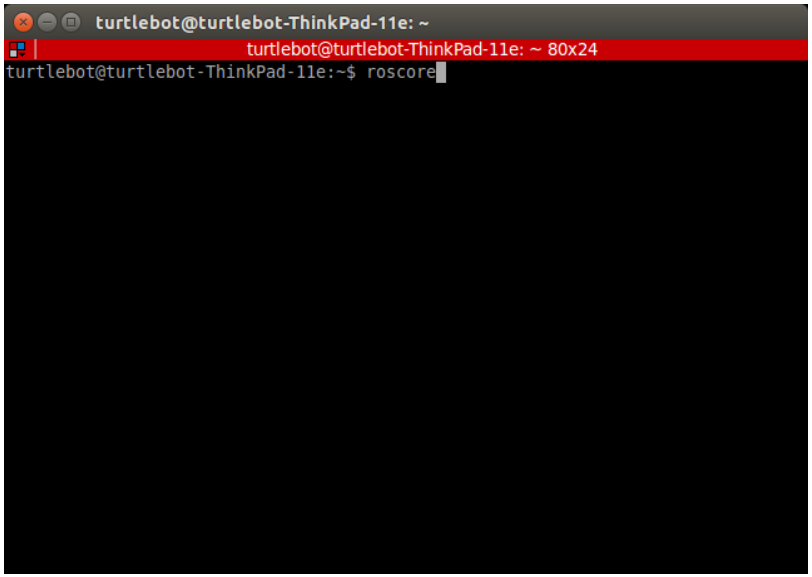
- `roscore`, also known as ROS Master, should be executed before all other ROS packages.
- `catkin_create_pkg` creates a package which will contain the user's nodes.
- `catkin_make` is the software that builds the user's ROS package. It also automatically generates the source code that enables IPC.
- `rosrun` executes an individual node.
- `roslaunch` executes a set of nodes.
  - The information of the nodes and their parameters are set in a special XML file.

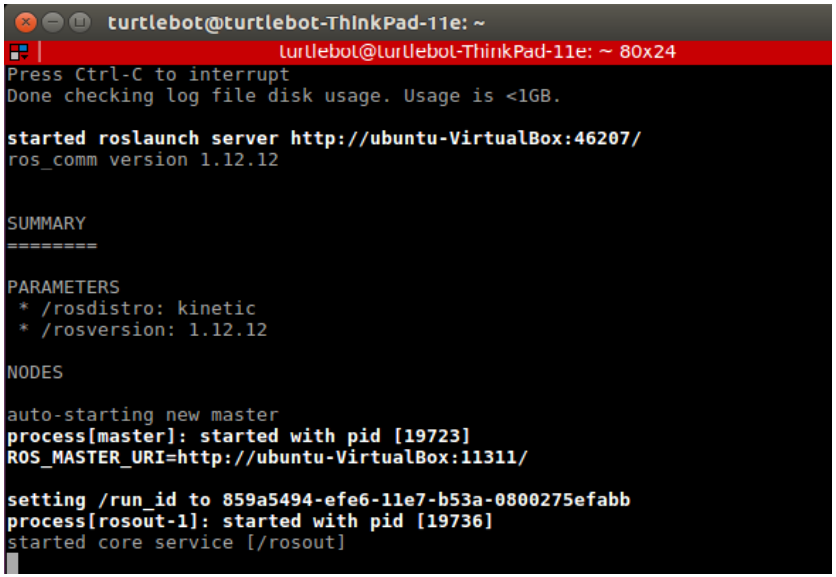
ROS command-line tools

- `rostopic` allows listing all the currently available topics, as well as printing the messages being published by them.
- `rosservice` allows listing and calling of all the currently available services.
- `rosmsg` is used to list and get information about the messages recognized by ROS.

roscore

- Usage:
  ```
  $ roscore
  ```
- Brings up the ROS Master
- Must be launched before all other nodes.

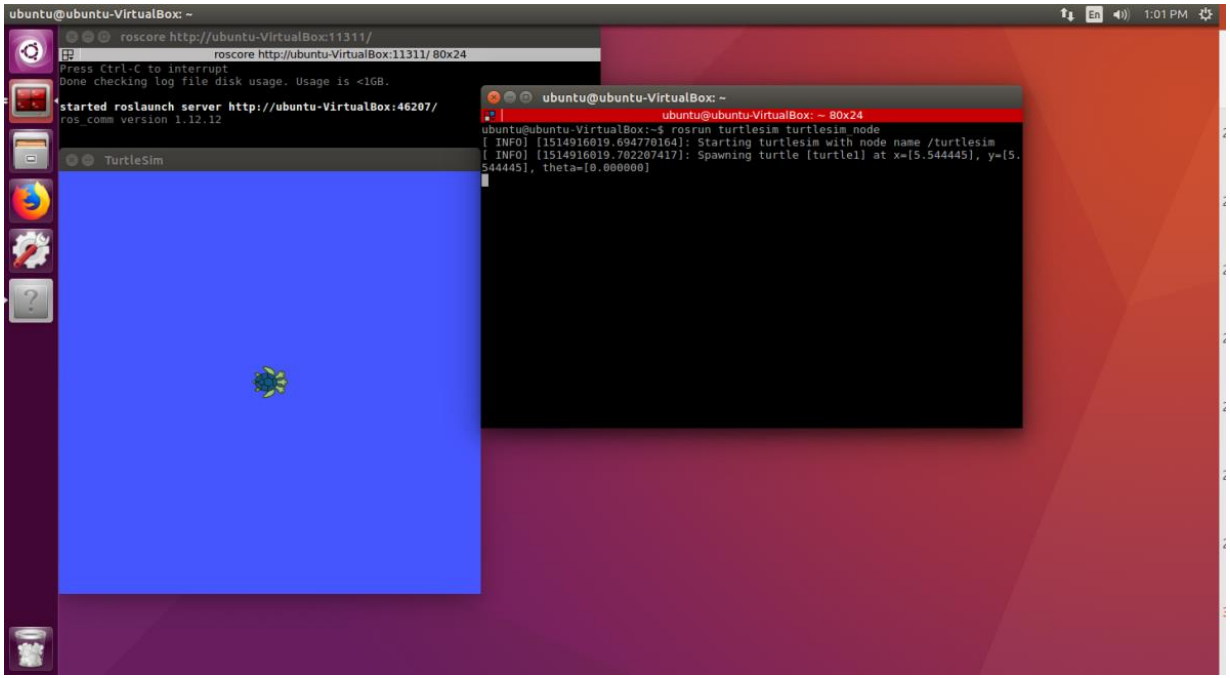




Reference: http://wiki.ros.org/roscore

rosrun

- Usage:
  ```
  $ rosrun <package> <package_node>
  ```
- Each node needs its own terminal.
- Example for TurtleSim node, a ROS tutorial introductory package:
  ```
  $ rosrun turtlesim turtlesim_node
  ```



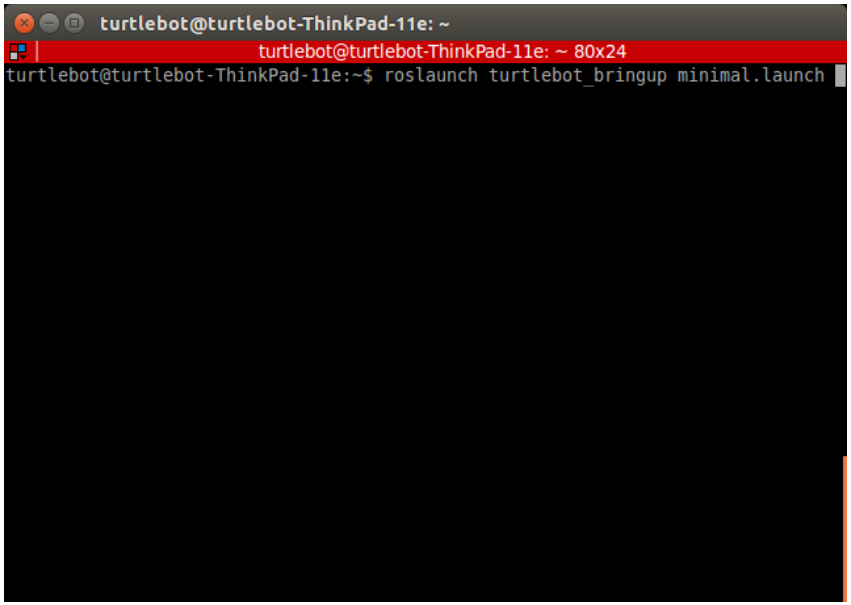
References:
http://wiki.ros.org/rosbash#rosrun
http://wiki.ros.org/turtlesim

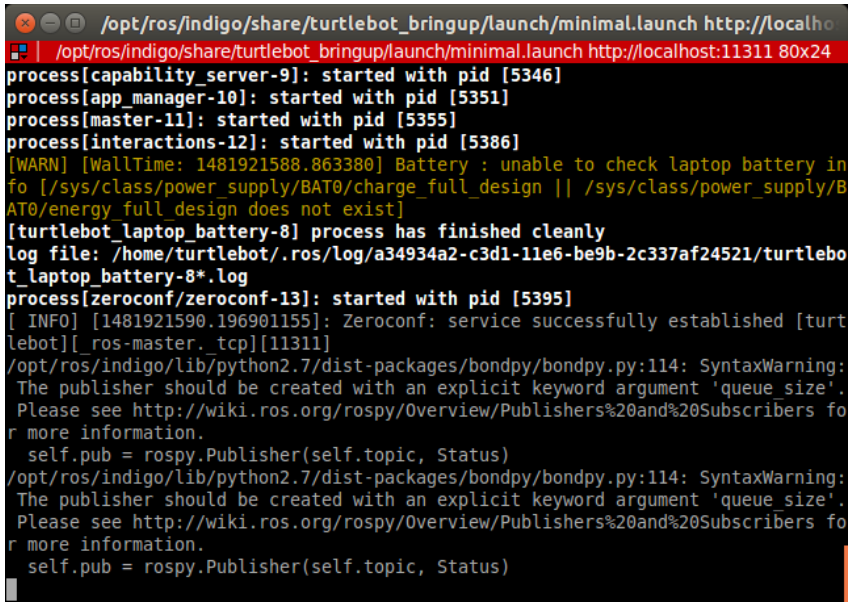
roslaunch

- Usage:

  ```
  $ roslaunch <package> <launch_file.launch>
  ```

- Initiates several nodes at the same time.
- Example for the Turtlebot launch file:





Reference: http://wiki.ros.org/roslaunch

rqt_graph

- GUI that allows you to see all the nodes that are currently running, and how they communicate with each other.
- Usage:

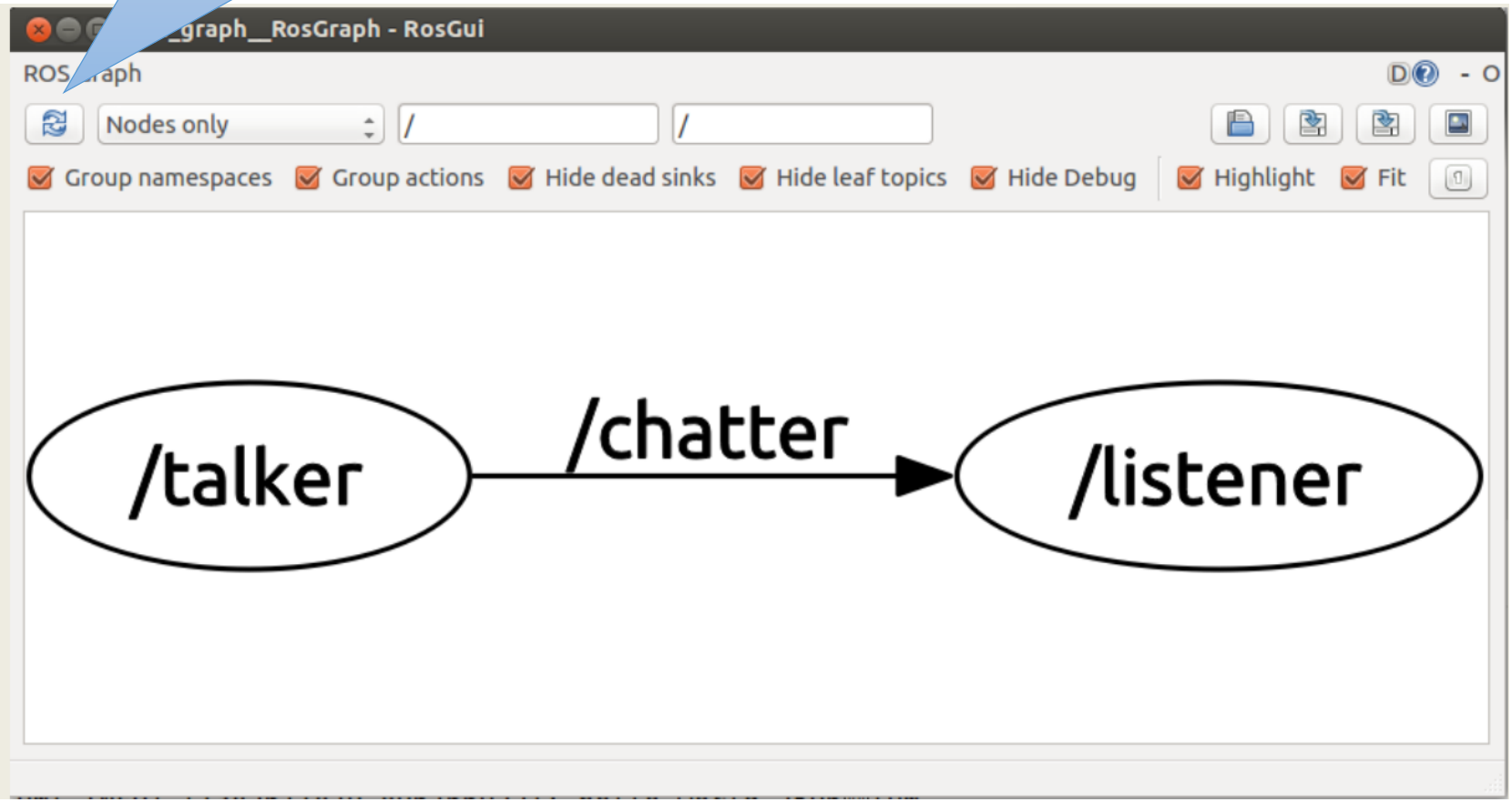
```
$ rqt_graph
```

- To show a topic publishing rate and average message age (delay), run this command before starting `rqt_graph`:

```
$ rosparam set enable_statistics true
```

rqt_graph

Updates the current view.

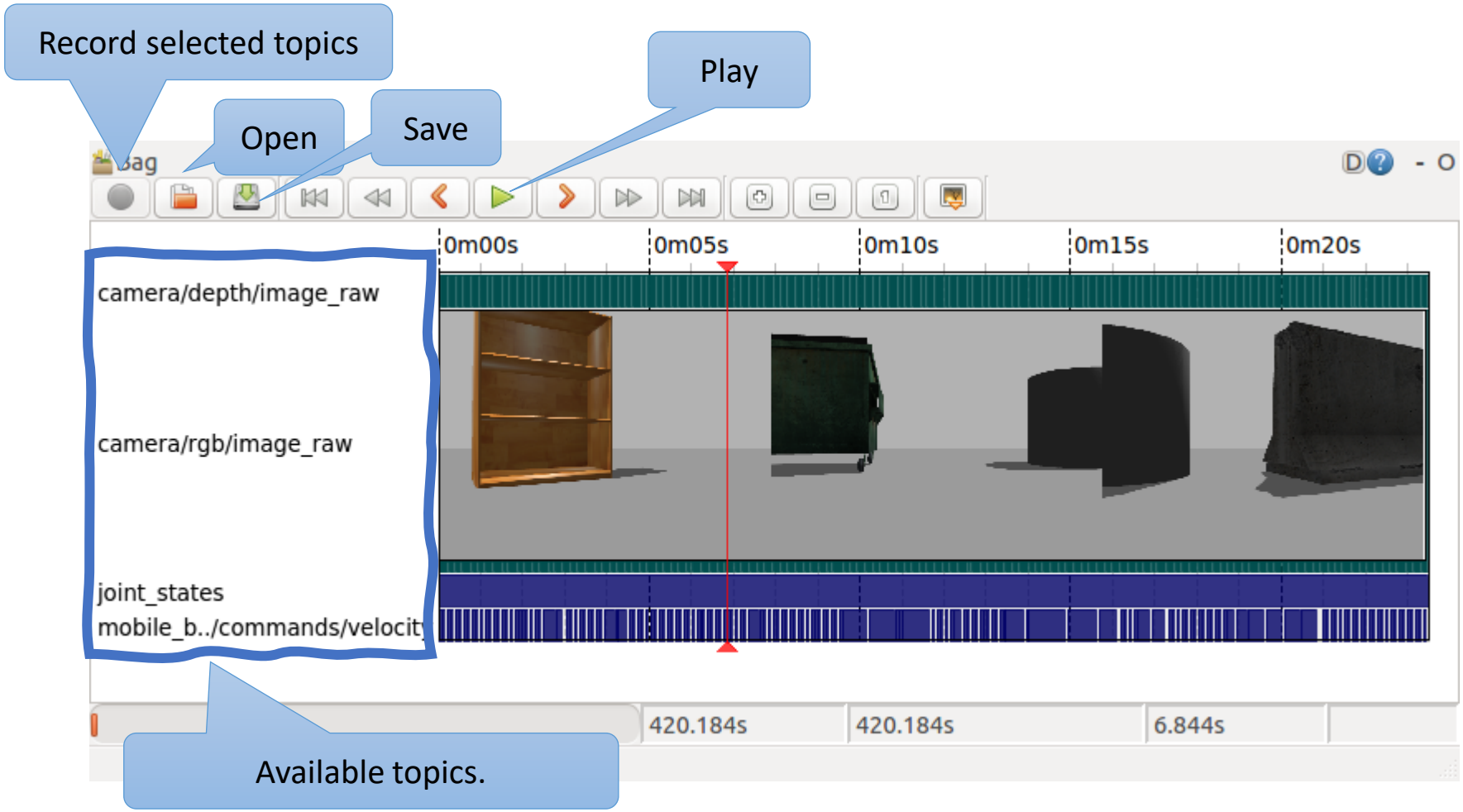


rqt_bag

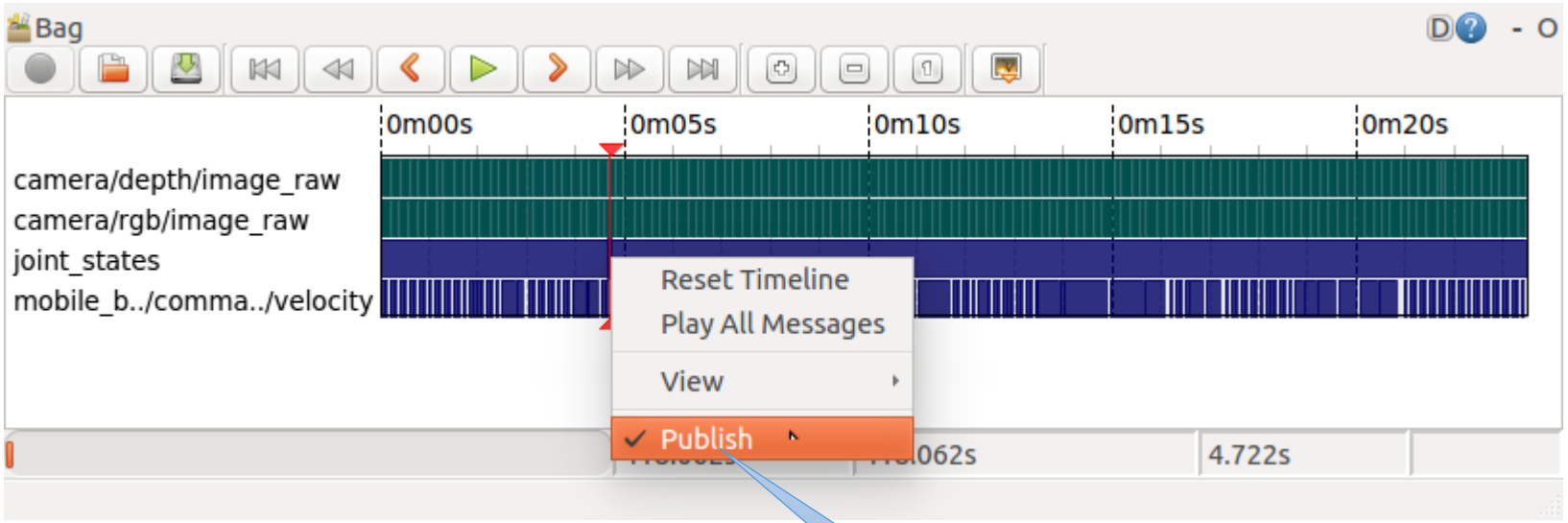
- GUI that allows you to record and/or reproduce messages being published by topics.
- The session is recorded in **`*.bag`** files.
- It is useful for **testing perception software** with recorded data.
  - Example: record the `/scan` messages to test the obstacle detection algorithm.
- **Warning**: bag files can become very large really fast.
- Usage:

```
$ rqt_bag
```

rqt_bag


Record selected topics
Open
Save
Play
bag
0m00s
0m05s
0m10s
0m15s
0m20s
camera/depth/image_raw
camera/rgb/image_raw
joint_states
mobile_b../commands/velocity
420.184s
420.184s
6.844s
Available topics.

rqt_bag



Right-click topic to enable/disable publishing

ROS References

- http://www.ros.org/
- http://wiki.ros.org/
- http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
- http://wiki.ros.org/ROS/Tutorials/CreatingPackage
- http://wiki.ros.org/Master
- http://wiki.ros.org/Nodes
- http://wiki.ros.org/Topics
- http://wiki.ros.org/Messages
- http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers
- http://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning
- http://wiki.ros.org/catkin/commands/catkin_make
- http://wiki.ros.org/roscore
- http://wiki.ros.org/rosbash#rosrun
- http://wiki.ros.org/roslaunch

ROS Official Tutorial

- If you want to learn more about ROS, in this course we suggest you read through the official tutorials found at: http://wiki.ros.org/ROS/Tutorials

- The official tutorials will guide you through the process of creating your own custom packages.

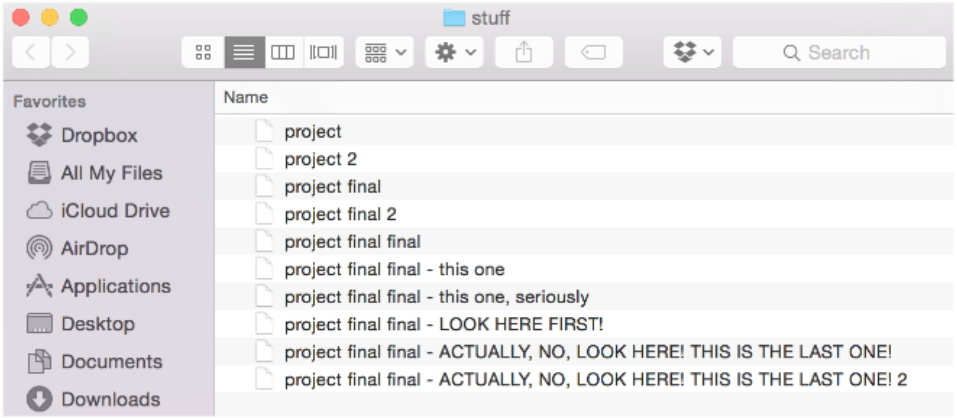
Using Git

How to work together

UNIVERSITY OF TORONTO

Git

- **Git** (/gɪt/) is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among among programmers collaboratively developing source code during software development.
- You can use git to work together on the same project.
- Helps you manage source files and keep history of your changes.
- You can work on the same file at the same time and combine your work through “merge” after you finished.
- You can have different branches of your work to try different things
- Avoid this:



DOGHOUSEDIARIES

Git

- Git is open source and is available by default on many Linux distros, you can have the repository on your computers or on the cloud.
- The most famous hosting service is: www.github.com
- Downloads:
  - GitHub Desktop: https://desktop.github.com/
  - Git for all platforms: https://git-scm.com/
- Quick Start: Quickstart - GitHub Docs

Git

- Useful Commands
  - $ git init
  - $ git remote add origin [url]
  - $ git clone [url]
  - $ git pull
  - $ git commit -m "[descriptive message]"
  - $ git push

init command turns an existing directory into a new Git repository

Specifies the remote repository for your local repository (Github).

Download a git repository from a remote location

Updates your current local working branch with all new commits from the remote branch

Records file snapshots permanently in version history

Uploads all local branch commits to GitHub

Usual workflow

Git References

- Quickstart - GitHub Docs

Virtual Machine and Docker

Getting started with Ubuntu and ROS

Pooya Poolad

Outline

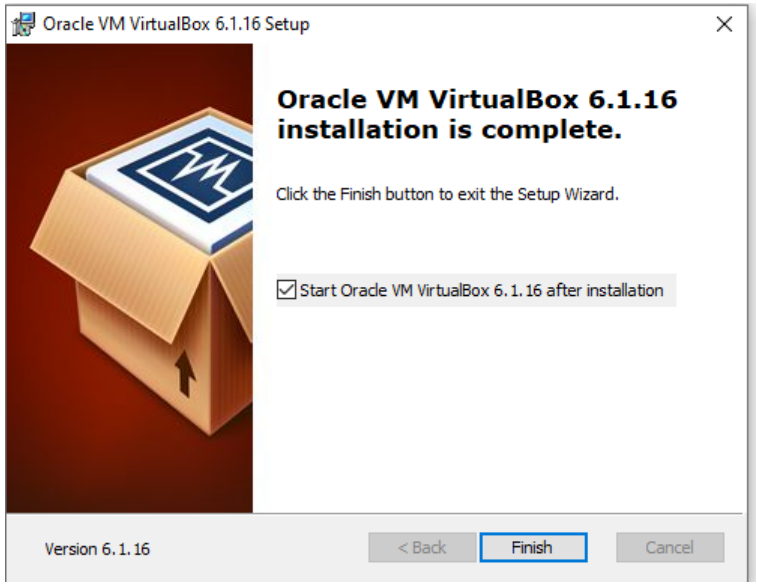
- Installing Ubuntu on a Virtual Machine
- Using Docker on Windows

Installing VM

- Oracle VirtualBox is a Free and powerful software:
  1. Download VirtualBox:
     - https://www.virtualbox.org/wiki/Downloads
  2. Download VirtualBox Extension pack (Make sure it matches your version; 7.0.4 is the latest):
     - https://download.virtualbox.org/virtualbox/7.0.4/Oracle_VM_VirtualBox_Extension_Pack-7.0.4.vbox-extpack

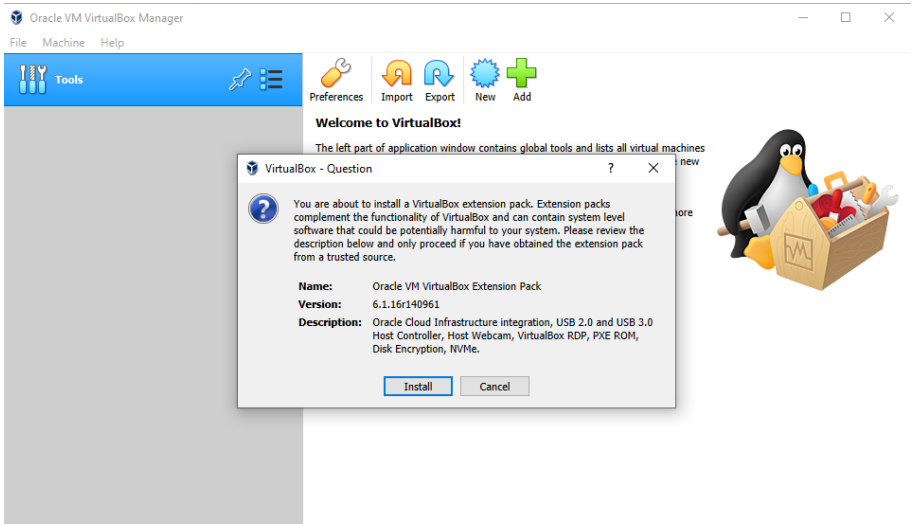
VirtualBox

- Installing VirtualBox is straight forward (Just hit next)
  - Make sure Network driver will be installed



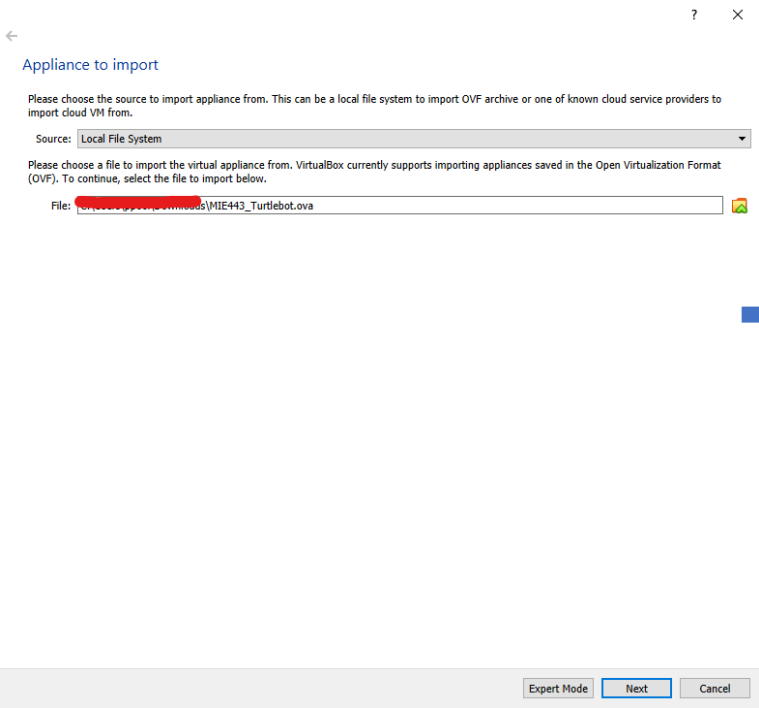
VirtualBox extension

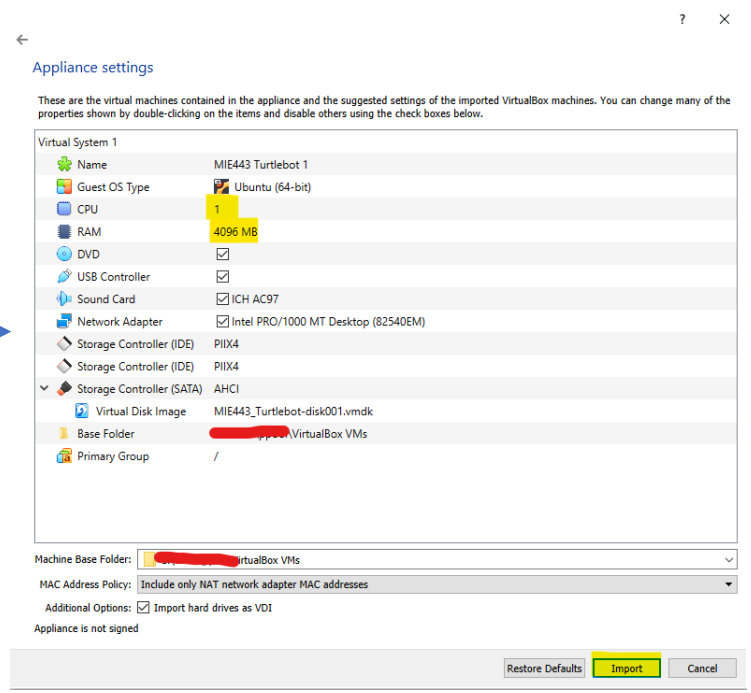
- Double click on the downloaded file
  1. VirtualBox should open
  2. Click on install



Turtlebot Image

- Download Turtlebot image (4.6GB) from:
  - [MIE443_Turtlebot.ova]

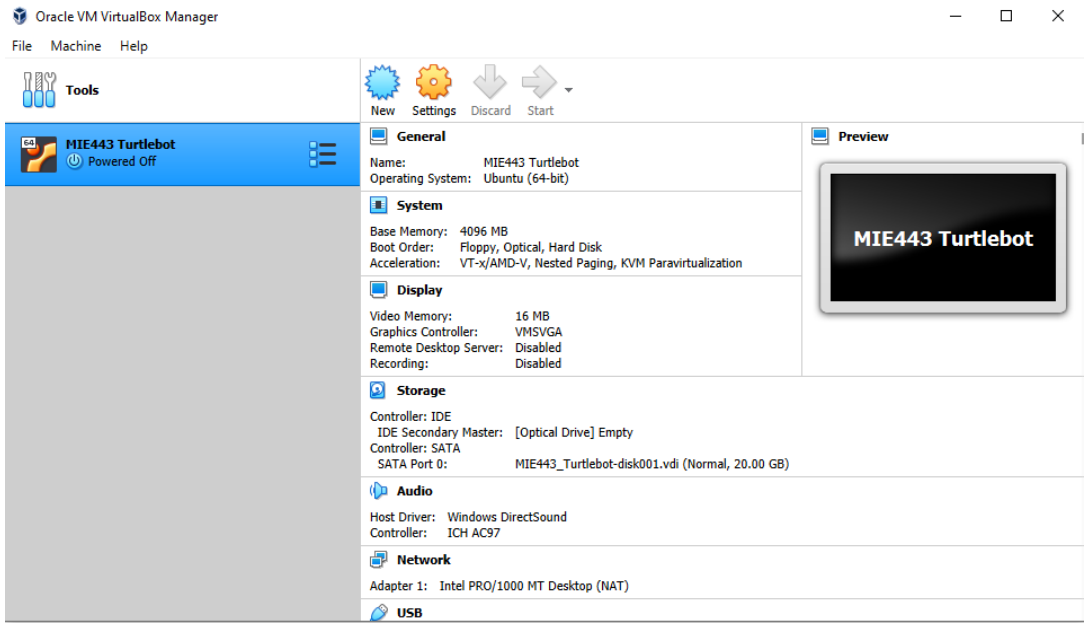




UNIVERSITY OF TORONTO

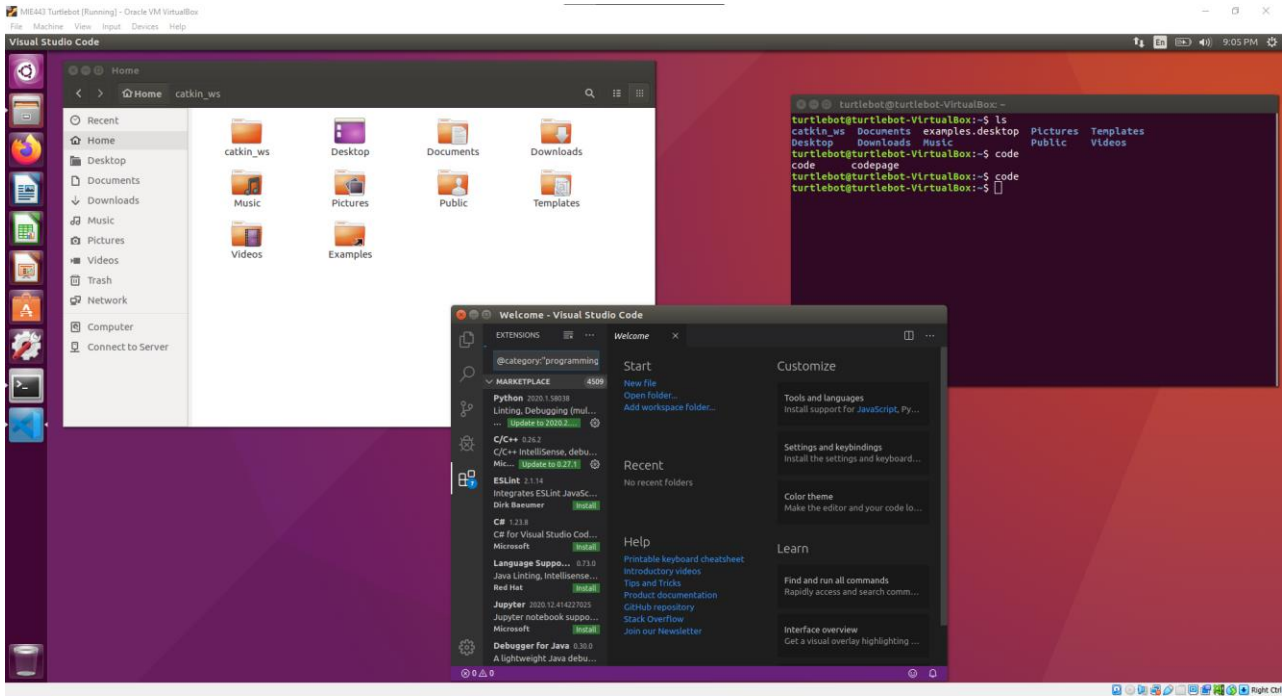
Imported!

- Right click and start VM.
- Password: turtlebot



Inside the VM

- Everything is installed.
- You can start exploring right away.



Update Gazebo

- If you ran into issues running simulations on Gazebo, try updating to version 7

```
sudo apt-get install wget
```

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

```
sudo apt update
```

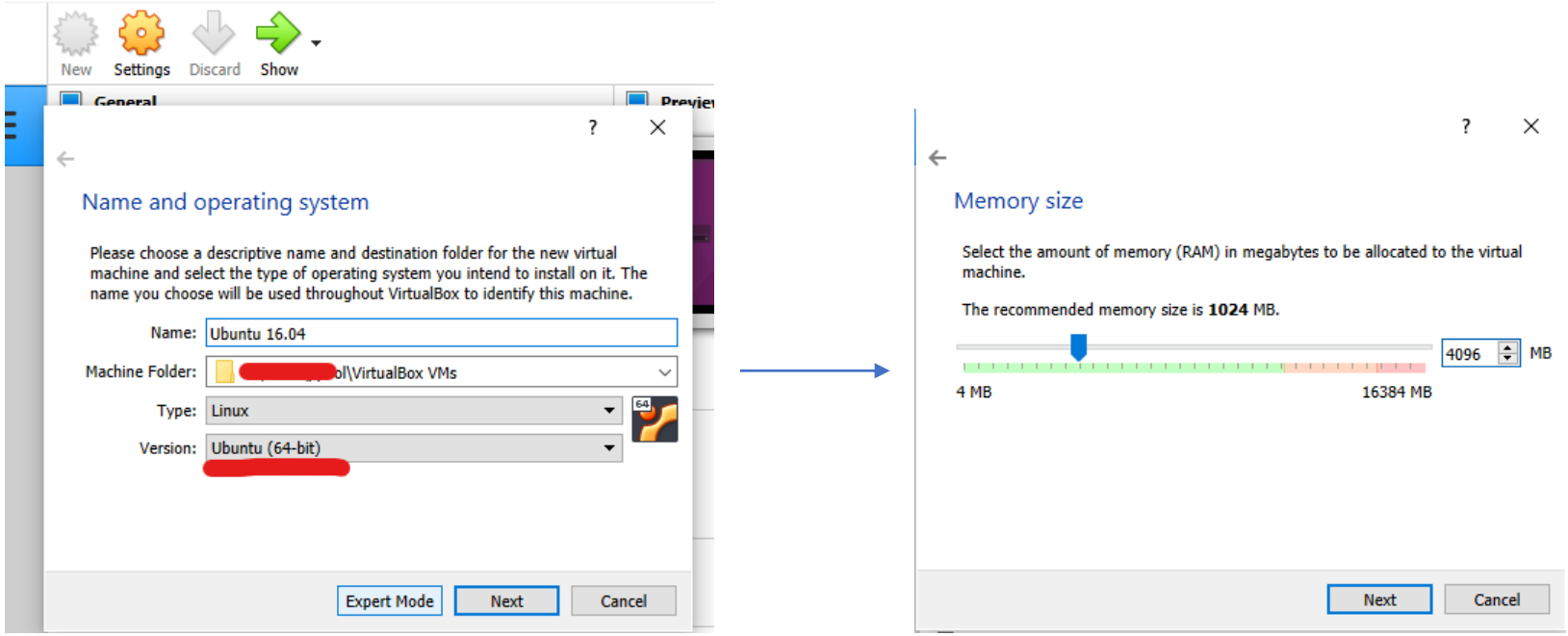
```
sudo apt upgrade
```

Virtual Machine From Scratch

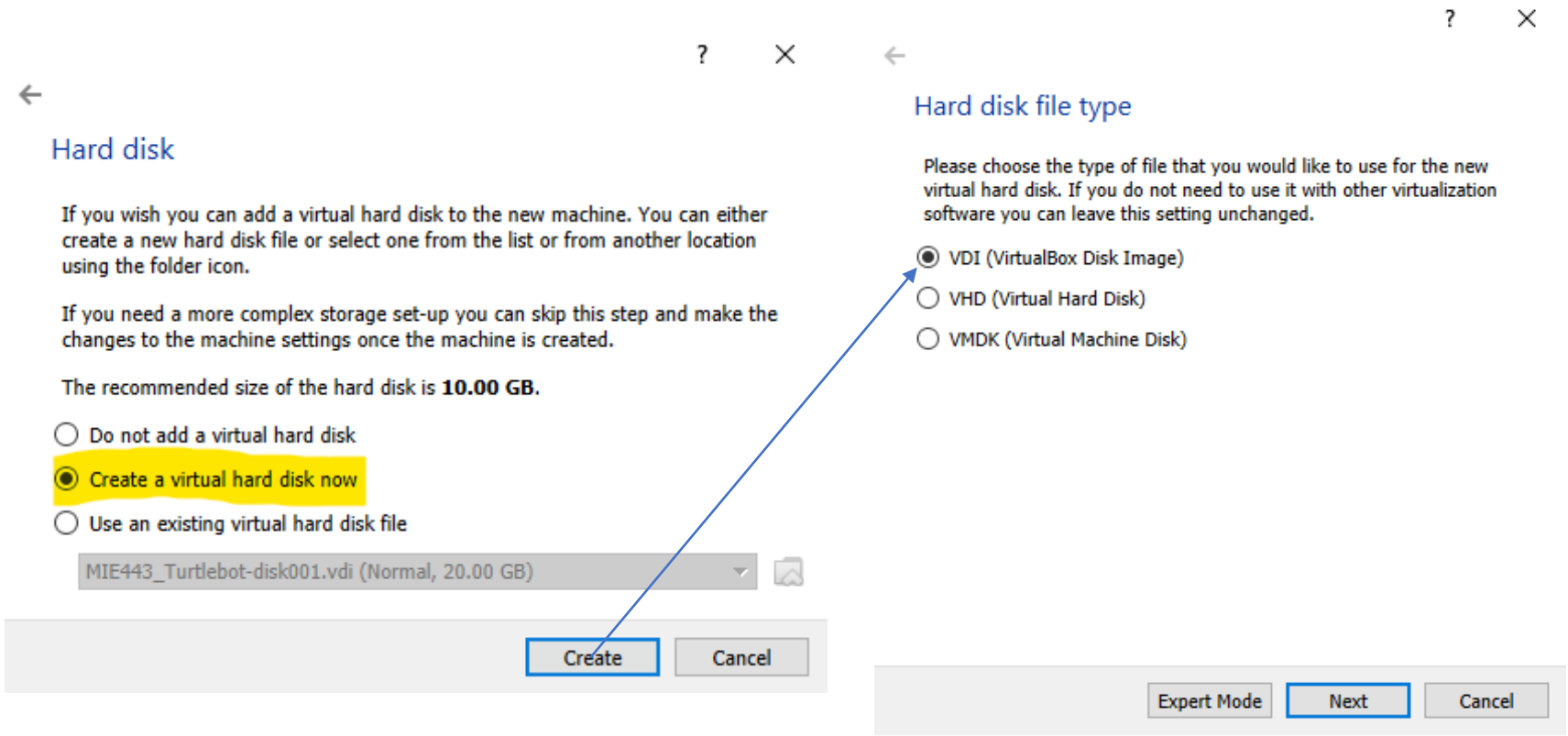
- Alternatively, you can create a VM from scratch
  1. Download Ubuntu 16.04 Image
     1. X64: http://releases.ubuntu.com/16.04.6/ubuntu-16.04.6-desktop-amd64.iso
     2. X86: http://releases.ubuntu.com/16.04.6/ubuntu-16.04.6-desktop-i386.iso
  2. Create new VM

Virtual Machine From Scratch

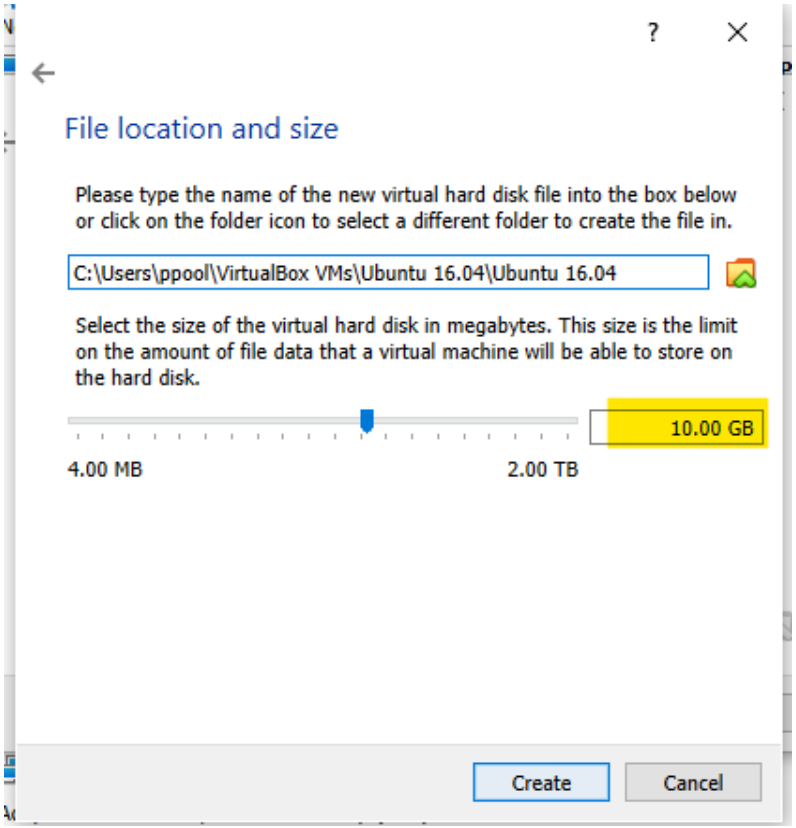
1. Create new VM



Create Disk

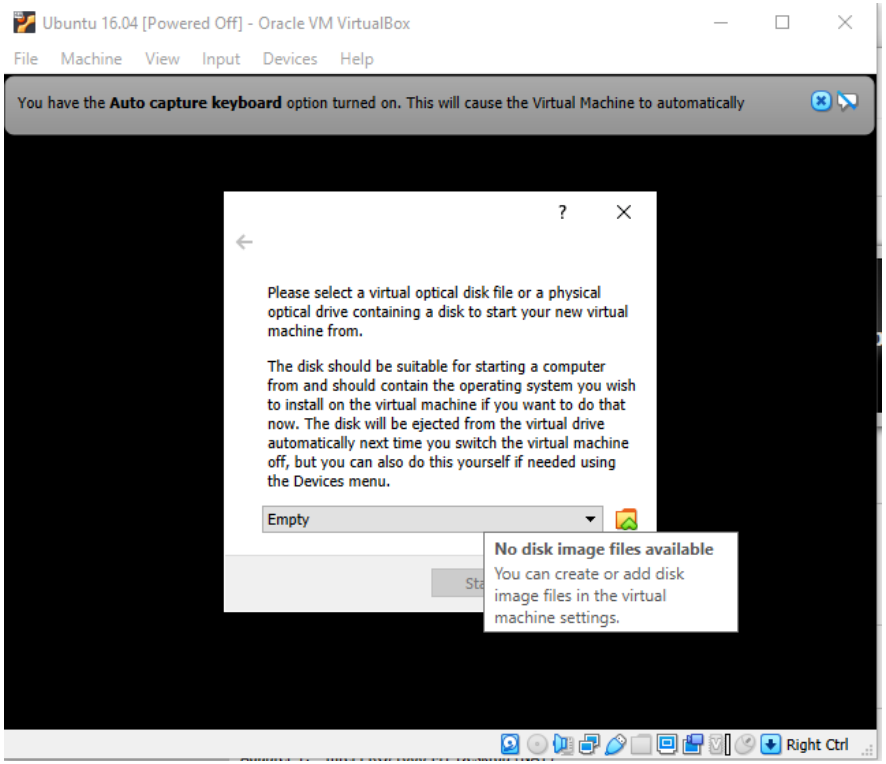

Hard disk
If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.
The recommended size of the hard disk is 10.00 GB.
Do not add a virtual hard disk
Create a virtual hard disk now
Use an existing virtual hard disk file
MIE443_Turtlebot-disk001.vdi (Normal, 20.00 GB)
Create
Cancel
Hard disk file type
Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.
VDI (VirtualBox Disk Image)
VHD (Virtual Hard Disk)
VMDK (Virtual Machine Disk)
Expert Mode
Next
Cancel

Allocate Space



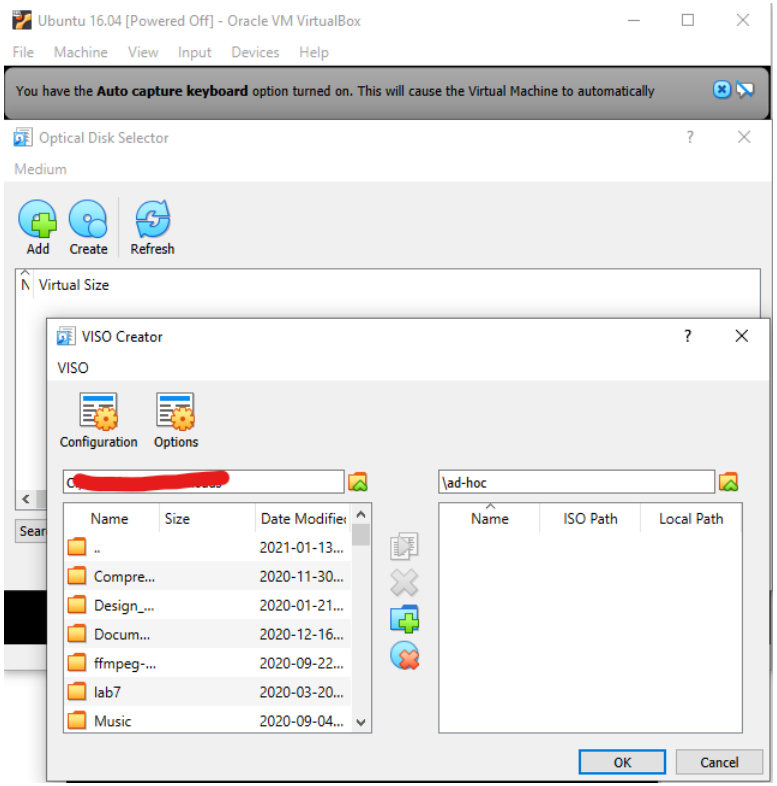
Install the OS

- It is an empty machine, you have to install an OS.



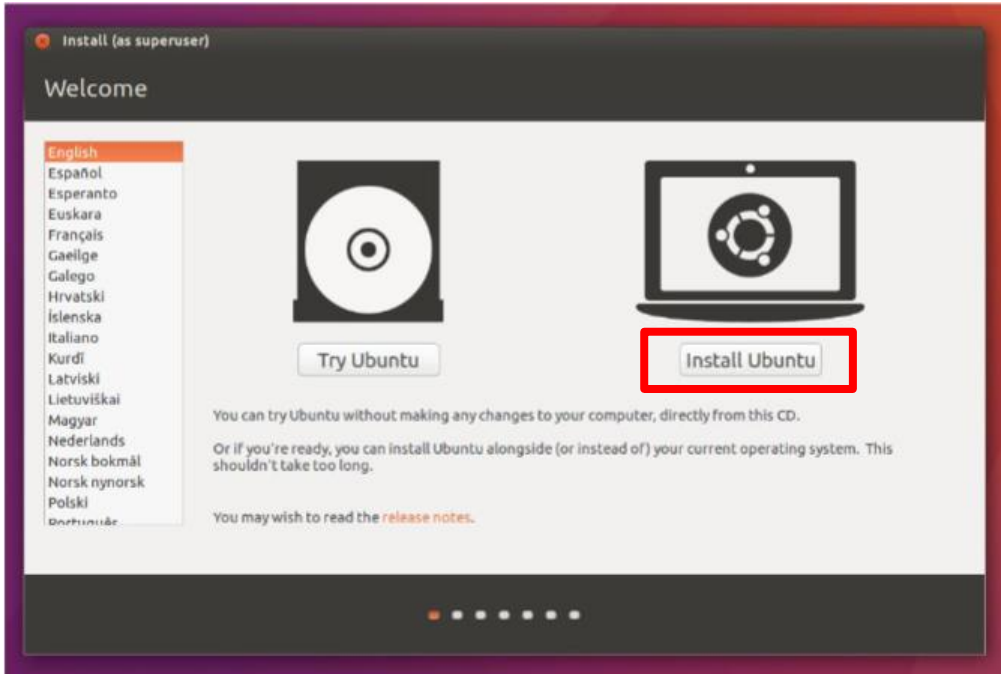
Import boot device

- Select and add the ISO you just downloaded as a boot device:



Install guest OS

- Install Ubuntu



Install ROS

- Install ROS & Turtlebot using the given script
  - Download the script and other given files
  - Open Terminal
  - > source /path/to/script/turtlebot_script.sh
  - If in Downloads: > source ~/Downloads/turtlebot_script.sh

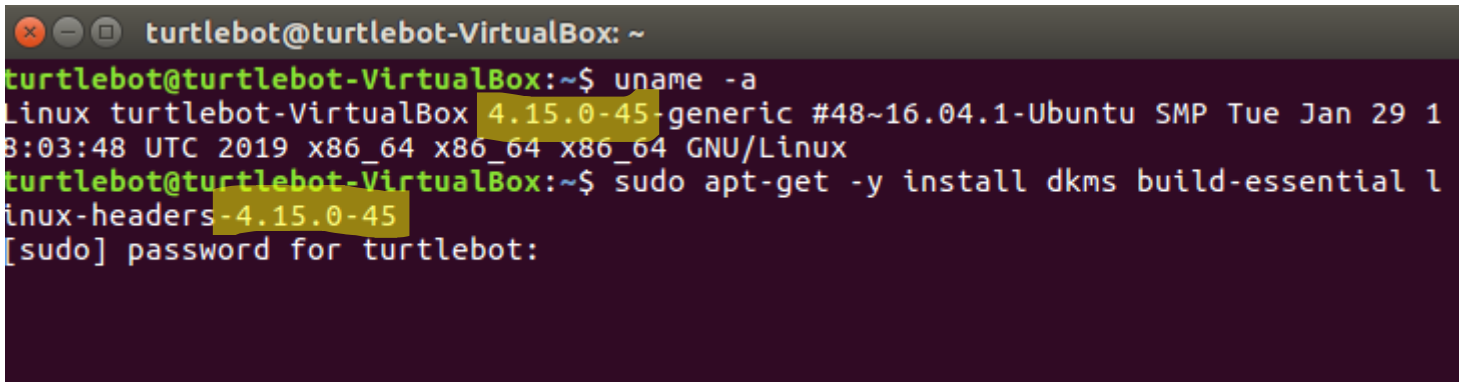
Install Guest Additions

- Optimizes the OS for better performance
  - Prepare linux by installing dependencies

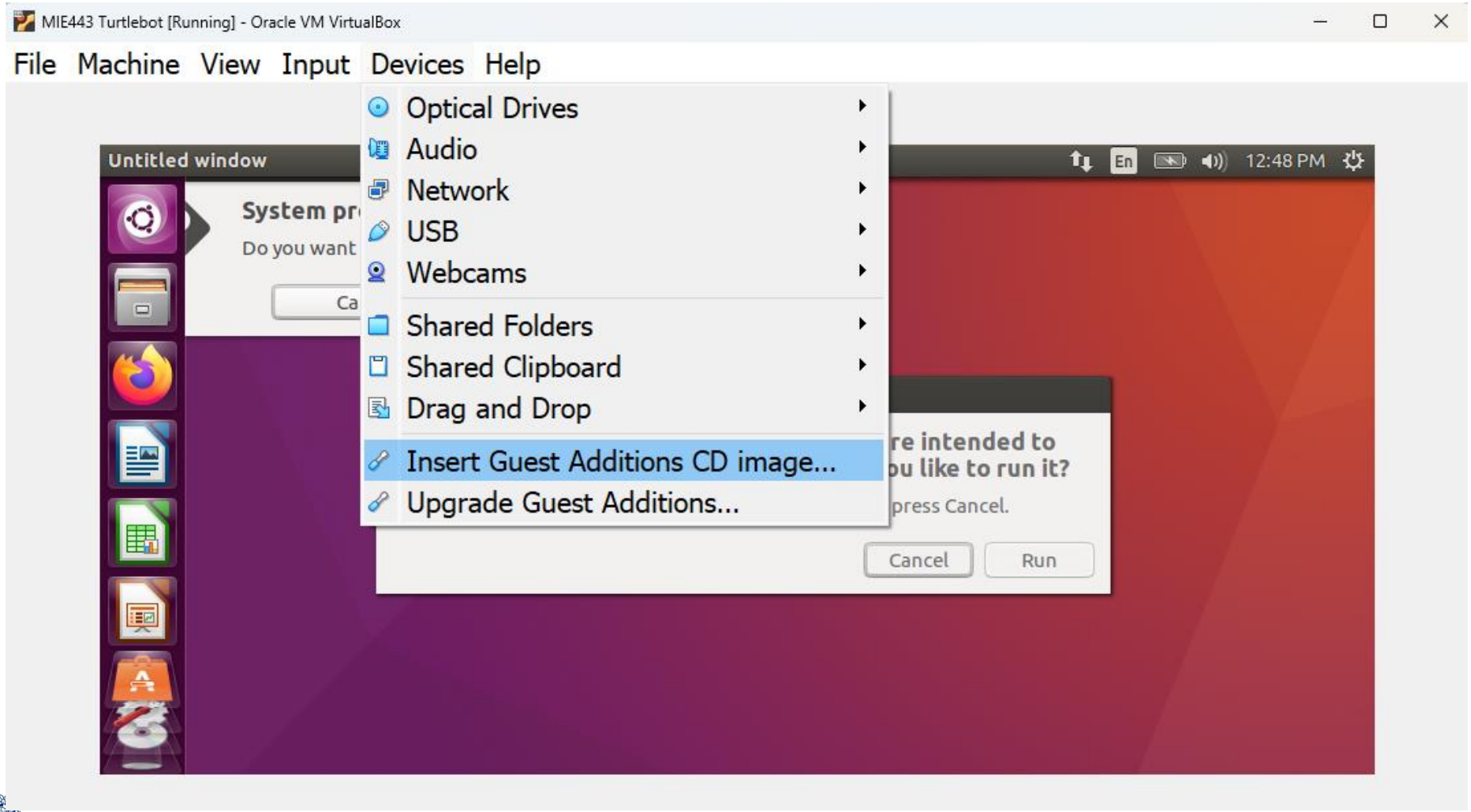
    $ uname –a #check version

    $ sudo apt-get -y install dkms build-essential linux-headers-VERSION #replace version with the one you get (the provided ova file's kernel version is 4.15.0-45

    $sudo reboot

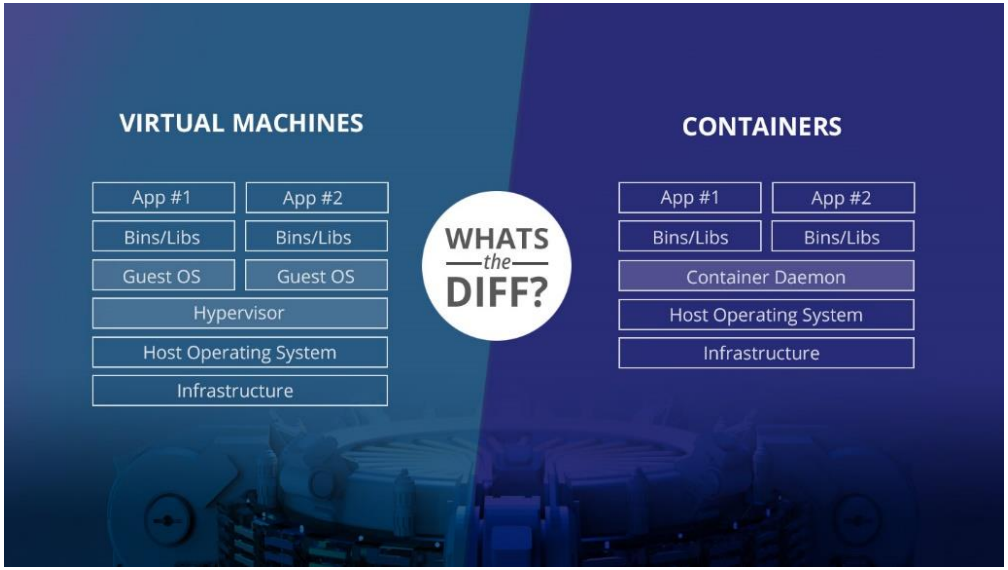


Install Guest Additions



Using Docker

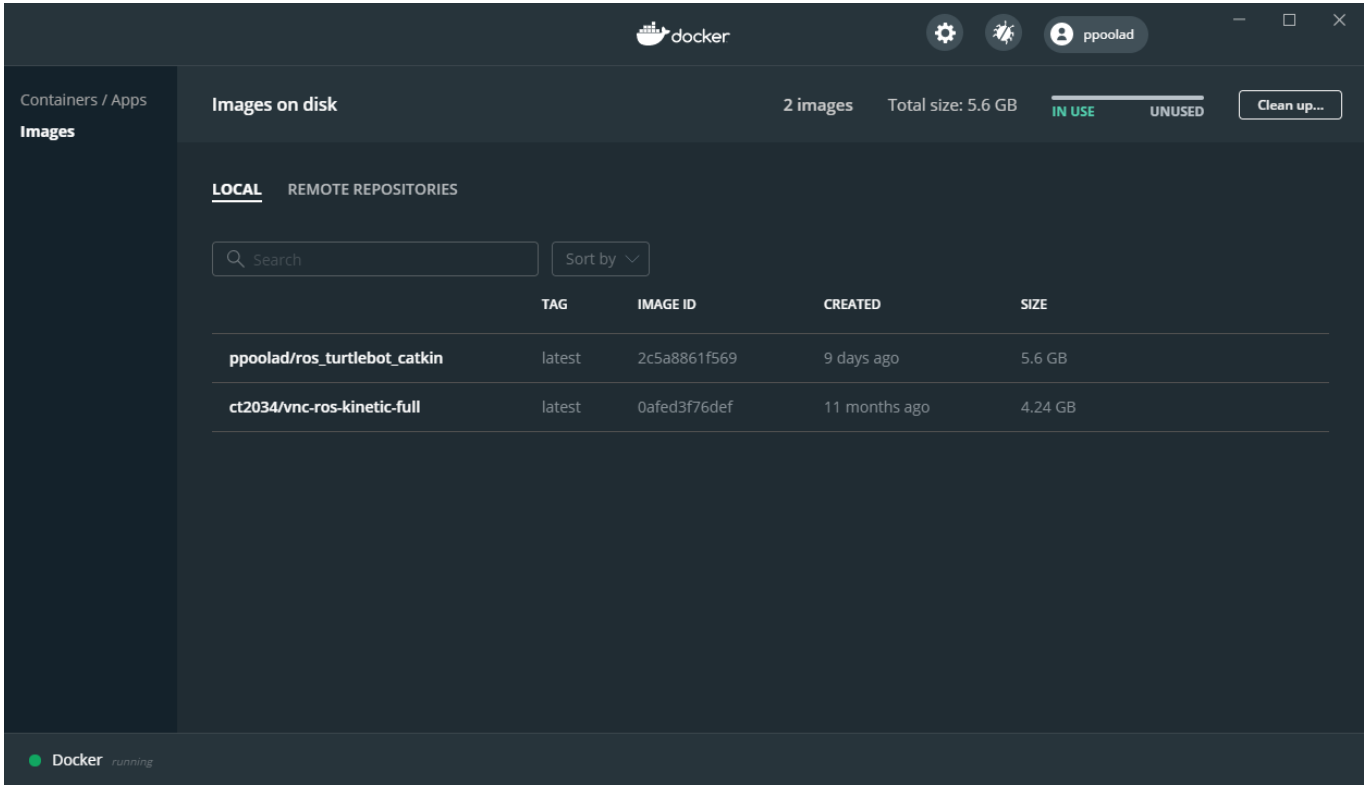
- Why?
- Do we need a whole VM?
  - Containers are a technology that provide us only what is necessary to run a certain application



Installing Docker

- Install Docker Desktop:
  - Get Started with Docker | Docker
- Restart PC if required
- Open a terminal (Powershell or WSL)
  - Pull the image we will be using:
    - > docker pull daaniiel/mie443-docker3
    - It takes couple of minutes to download and setup
  - Run the Image:
    - > docker run -v <your directory>:/mnt/ -it --rm -p 6080:80 -p 5900:5900 ct2034/vnc-ros-kinetic-full

Docker Desktop



Run a Container

```
ppoolad@SurfaceBook-Pooya:/mnt/c/Users/ppool$ docker run -v ~/MIE443:/mnt/MIE443 -it --rm -p 6080:80 -p 5900:590
0 ct2034/vnc-ros-kinetic-full
/usr/lib/python2.7/dist-packages/supervisor/options.py:297: UserWarning: Supervisord is running as root and it i
s searching for its configuration file in default locations (including its current working directory); you proba
bly want to specify a "-c" argument specifying an absolute path to a configuration file for improved security.
  'Supervisord is running as root and it is searching '
2021-01-14 03:44:08,967 CRIT Supervisor running as root (no user in config file)
2021-01-14 03:44:08,967 WARN Included extra file "/etc/supervisor/conf.d/supervisord.conf" during parsing
2021-01-14 03:44:08,984 INFO RPC interface 'supervisor' initialized
2021-01-14 03:44:08,984 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2021-01-14 03:44:08,984 INFO supervisord started with pid 23
2021-01-14 03:44:09,987 INFO spawned: 'xvfb' with pid 27
2021-01-14 03:44:09,988 INFO spawned: 'pcmanfm' with pid 28
2021-01-14 03:44:09,991 INFO spawned: 'lxpanel' with pid 29
2021-01-14 03:44:09,993 INFO spawned: 'lxsession' with pid 30
2021-01-14 03:44:09,995 INFO spawned: 'x11vnc' with pid 31
2021-01-14 03:44:09,999 INFO spawned: 'novnc' with pid 32
2021-01-14 03:44:11,029 INFO success: xvfb entered RUNNING state, process has stayed up for > than 1 seconds (st
artsecs)
2021-01-14 03:44:11,029 INFO success: pcmanfm entered RUNNING state, process has stayed up for > than 1 seconds
(startsecs)
2021-01-14 03:44:11,029 INFO success: lxpanel entered RUNNING state, process has stayed up for > than 1 seconds
(startsecs)
2021-01-14 03:44:11,029 INFO success: lxsession entered RUNNING state, process has stayed up for > than 1 second
s (startsecs)
2021-01-14 03:44:11,029 INFO success: x11vnc entered RUNNING state, process has stayed up for > than 1 seconds (
startsecs)
2021-01-14 03:44:11,029 INFO success: novnc entered RUNNING state, process has stayed up for > than 1 seconds (s
tartsecs)
```

Run a container

- Using VNCviewer on port 5900
- Opening browser on localhost:6080

Setup Catkin

- To have catkin folder set up run modified script “Turtlebot_script_catkinws_setup.sh”
  - Open Terminator
  - > source /path/to/script.sh

Notes

- Docker containers are not the same as VMs
  - **Docker container will be reset to the initial state after closing it**
    - If you modify something in the system, you need to save the state via docker commit
    - That is why mounting is important
  - Always save your file on host side (They will be deleted)!
  - VM is the suggested way for normal users.

VM References

- Oracle® VM VirtualBox®
- Docker Documentation | Docker Documentation
- Install WSL | Microsoft Learn