Lab 01
Programming Basics

Song Chaoyang

Assistant Professor

Department of Mechanical and Energy Engineering

songcy@sustc.edu.cn

Agenda

Week 01, Friday, February 22

- Meet the Robots
- Intro to Linux
- Programing Robot: C++ and Python
- ROS Introduction

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Meet the Robots

To Use Until Project 3

- Robot Arms + Grippers
  - AUBO i5 + Suction Cup
  - Universal Robots UR5 + Suction Cup
  - Universal Robots UR10e + Robotiq Hand-E
  - Franka Emika + Built-in 2 Finger Gripper
- Computers: Intel NUC with Linux & ROS installed
- Cameras: Intel RealSense D435
- Robot Station
- Safety Rules

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Linux

Best operating system for robotics development

- Ease of access and usage for abundant open source tools in Robotics community.

✓Communication method between multiple hardware or application processes

✓Simulation tools

✓Image Processing

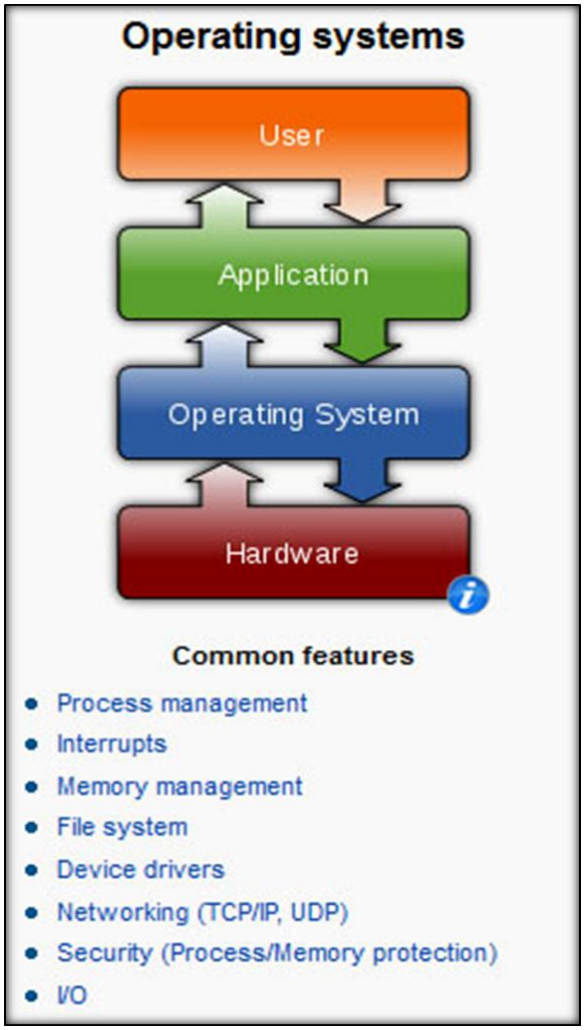
✓GPU computation

✓AI or Machine learning packages

AncoraSIR.com

Linux as a Operating System

- An operating system
  - like Windows, MacOS
- A Unix clone written from scratch
  - by Linus Torvalds.
  - Unix is the first multitasking, multi-user computer operating system.
- Widely adopted
  - 64% of the world's servers run some variant of Unix or Linux.
  - The Android phone and the Kindle run on Linux.



AncoraSIR.com

Linux Has Many Distributions

Which one to choose … a *big* question

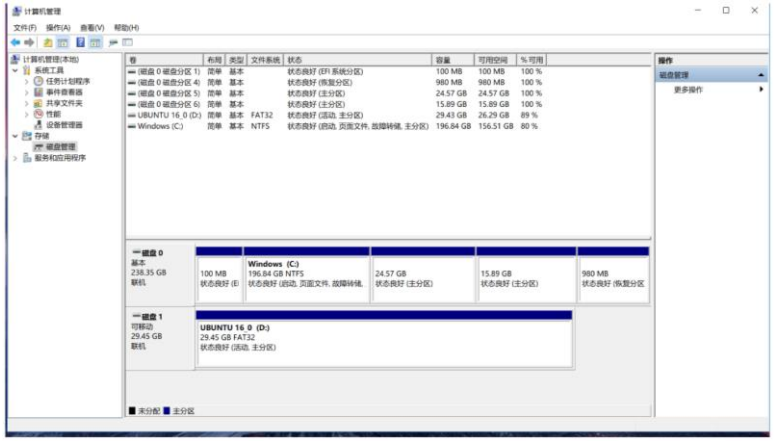
AncoraSIR.com

Ubuntu 16.04 Installation on a PC

Not the latest version, but stable to use and develop

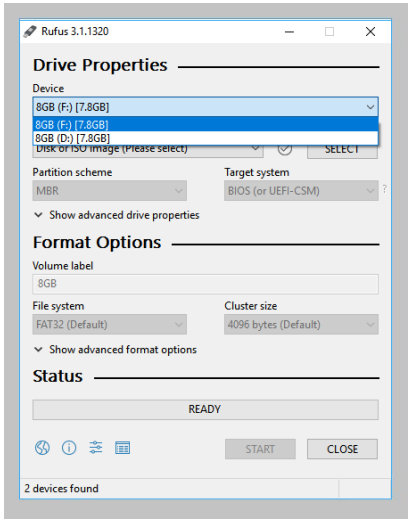
- Prepare your computer
  - Allocate disk space on your PC
  - Also available for other OSs

  https://github.com/ancorasir/BionicDL-CobotLearning-Project1/blob/master/Ubuntu%20installation%20tutorials.pdf



- Prepare the installation file
  - Create a bootable USB stick on Windows

  https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-windows#0



- Start the installation

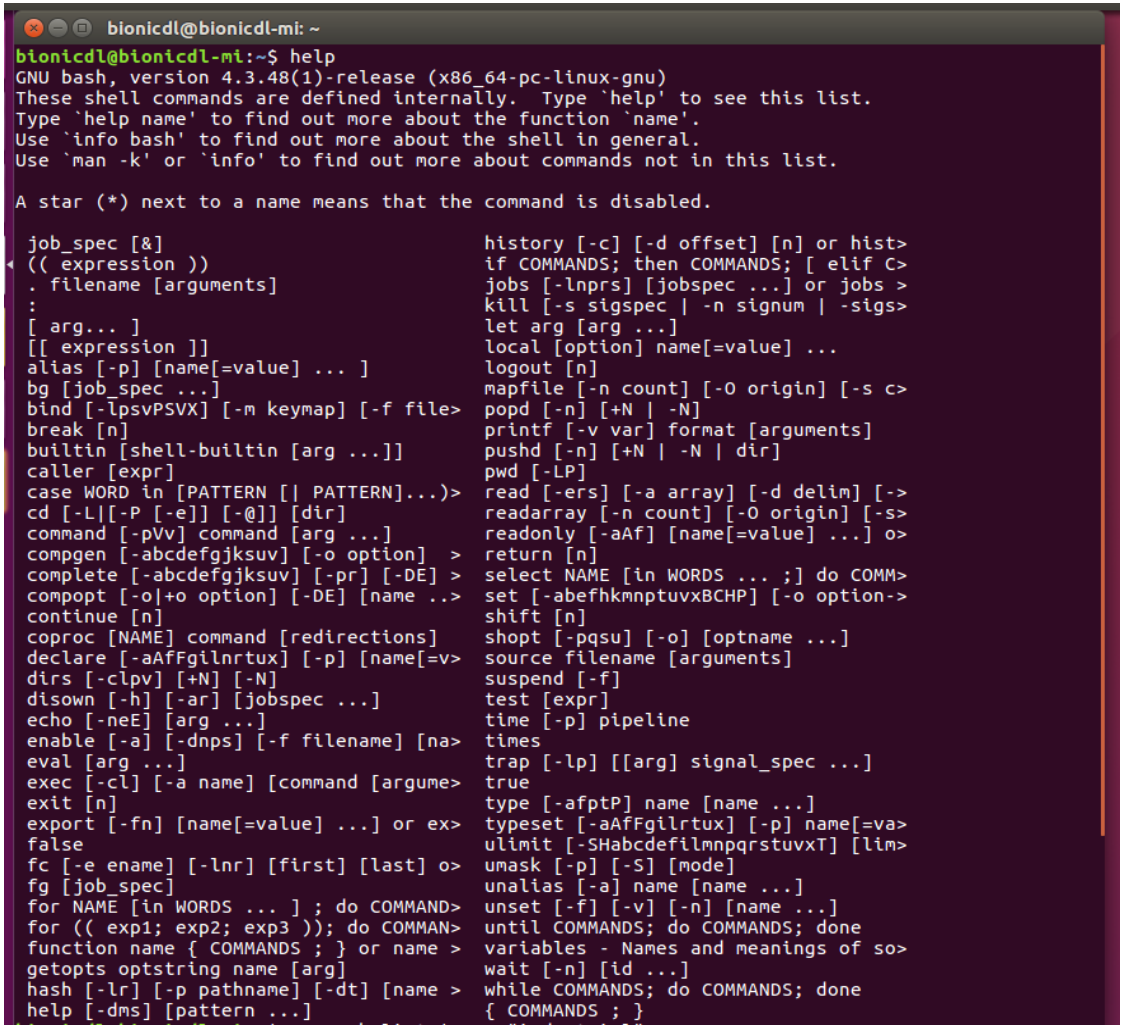
  https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop-1604#0

AncoraSIR.com

Get Familiar with Command line Tools

Small programs that do one thing well

- **Shells**:
  - BASH,
  - watch,
  - clear,
  - history,
  - echo,
  - Set
- *Practice, practice and practice …*



AncoraSIR.com

SUSTech
Southern University of Science and Technology

fournova

COMMAND LINE CHEAT SHEET

presented by **TOWER** › Version control with Git - made easy

DIRECTORIES

`$ pwd`

Display path of current working directory

`$ cd <directory>`

Change directory to <directory>

`$ cd ..`

Navigate to parent directory

`$ ls`

List directory contents

`$ ls -la`

List detailed directory contents, including hidden files

`$ mkdir <directory>`

Create new directory named <directory>

OUTPUT

`$ cat <file>`

Output the contents of <file>

`$ less <file>`

Output the contents of <file> using the less command (which supports pagination etc.)

`$ head <file>`

Output the first 10 lines of <file>

`$ <cmd> > <file>`

Direct the output of <cmd> into <file>

`$ <cmd> >> <file>`

Append the output of <cmd> to <file>

`$ <cmd1> | <cmd2>`

Direct the output of <cmd1> to <cmd2>

`$ clear`

Clear the command line window

FILES

`$ rm <file>`

Delete <file>

`$ rm -r <directory>`

Delete <directory>

`$ rm -f <file>`

Force-delete <file> (add `-r` to force-delete a directory)

`$ mv <file-old> <file-new>`

Rename <file-old> to <file-new>

`$ mv <file> <directory>`

Move <file> to <directory> (possibly overwriting an existing file)

`$ cp <file> <directory>`

Copy <file> to <directory> (possibly overwriting an existing file)

`$ cp -r <directory1> <directory2>`

Copy <directory1> and its contents to <directory2> (possibly overwriting files in an existing directory)

`$ touch <file>`

Update file access & modification time (and create <file> if it doesn't exist)

PERMISSIONS

`$ chmod 755 <file>`

Change permissions of <file> to 755

`$ chmod -R 600 <directory>`

Change permissions of <directory> (and its contents) to 600

`$ chown <user>:<group> <file>`

Change ownership of <file> to <user> and <group> (add `-R` to include a directory's contents)

SEARCH

`$ find <dir> -name "<file>"`

Find all files named <file> inside <dir> (use wildcards [*] to search for parts of filenames, e.g. "file.*")

`$ grep "<text>" <file>`

Output all occurrences of <text> inside <file> (add `-i` for case-insensitivity)

`$ grep -rl "<text>" <dir>`

Search for all files containing <text> inside <dir>

NETWORK

`$ ping <host>`

Ping <host> and display status

`$ whois <domain>`

Output whois information for <domain>

`$ curl -O <url/to/file>`

Download <file> (via HTTP[S] or FTP)

`$ ssh <username>@<host>`

Establish an SSH connection to <host> with user <username>

`$ scp <file> <user>@<host>:/remote/path`

Copy <file> to a remote <host>

PROCESSES

`$ ps ax`

Output currently running processes

`$ top`

Display live information about currently running processes

`$ kill <pid>`

Quit process with ID <pid>

fournova

COMMAND LINE TIPS & TRICKS

presented by **TOWER** › Version control with Git - made easy

GETTING HELP

On the command line, help is always at hand: you can either type `man <command>` or `<command> --help` to receive detailed documentation about the command in question.

FILE PERMISSIONS

On Unix systems, file permissions are set using three digits: the first one representing the permissions for the owning user, the second one for its group, and the third one for anyone else.

Add up the desired access rights for each digit as following:

`4` – access/read (r)
`2` – modify/write (w)
`1` – execute (x)

For example, `755` means "rwx" for owner and "rx" for both group and anyone. `740` represents "rwx" for owner, "r" for group and no rights for other users.

COMBINING COMMANDS

If you plan to run a series of commands after another, it might be useful to combine them instead of waiting for each command to finish before typing the next one. To do so, simply separate the commands with a semicolon (`;`) on the same line.

Additionally, it is possble to execute a command only if its predecessor produces a certain result. Code placed after the `&&` operator will only be run if the previous command completes successfully, while the opposite `||` operator only continues if the previous command fails. The following command will create the folder "videos" only if the `cd` command fails (and the folder therefore doesn't exist):

`$ cd ~/videos || mkdir ~/videos`

THE "CTRL" KEY

Various keyboard shortcuts can assist you when entering text: Hitting `CTRL+A` moves the caret to the beginning and `CTRL+E` to the end of the line.

In a similar fashion, `CTRL+K` deletes all characters after and `CTRL+U` all characters in front of the caret.

Pressing `CTRL+L` clears the screen (similarly to the `clear` command). If you should ever want to abort a running command, `CTRL+C` will cancel it.

THE "TAB" KEY

Whenever entering paths and file names, the `TAB` key comes in very handy. It autocompletes what you've written, reducing typos quite efficiently. E.g. when you want to switch to a different directory, you can either type every component of the path by hand:

`$ cd ~/projects/acmedesign/docs/`

...or use the `TAB` key (try this yourself):

`$ cd ~/pr[TAB]ojects/ac[TAB]medesign/d[TAB]ocs/`

In case your typed characters are ambiguous (because "ac" could point to the "acmedesign" or the "actionscript" folder), the command line won't be able to autocomplete. In that case, you can hit `TAB` twice to view all possible matches and then type a few more characters.

THE ARROW KEYS

The command line keeps a history of the most recent commands you executed. By pressing the `ARROW UP` key, you can step through the last called commands (starting with the most recent). `ARROW DOWN` will move forward in history towards the most recent call.

Bonus tip: Calling the `history` command prints a list of all recent commands.

HOME FOLDER

File and directory paths can get long and awkward. If you're addressing a path inside of your home folder though, you can make things easier by using the `~` character. So instead of writing `cd /Users/your-username/projects/` , a simple `cd ~/projects/` will do.

And in case you should forget your user name, `whoami` will remind you.

OUTPUT WITH "LESS"

The `less` command can display *and paginate* output. This means that it only displays one page full of content and then waits for your explicit instructions. You'll know you have `less` in front of you if the last line of your screen either shows the file's name or just a colon (`:`).

Apart from the arrow keys, hitting `SPACE` will scroll one page forward, `b` will scroll one page backward, and `q` will quit the `less` program.

DIRECTING OUTPUT

The output of a command does not necessarily have to be printed to the command line. Instead, you can decide to direct it to somewhere else.

Using the `>` operator, for example, output can be directed to a file. The following command will save the running processes to a text file in your home folder:

`$ ps ax > ~/processes.txt`

It is also possible to pass output to another command using the `|` (pipe) operator, which makes it very easy to create complex operations. E.g., this chain of commands will list the current directory's contents, search the list for PDF files and display the results with the `less` command:

`$ ls | grep ".pdf" | less`

30-day free trial available at
www.git-tower.com

TOWER
Version control with Git - made easy

Robot Programing

Top 2 popular programming languages in Robotics

How many of you are familiar with C++ or Python?

AncoraSIR.com

C/C++ vs. Python

In general and for robot programming

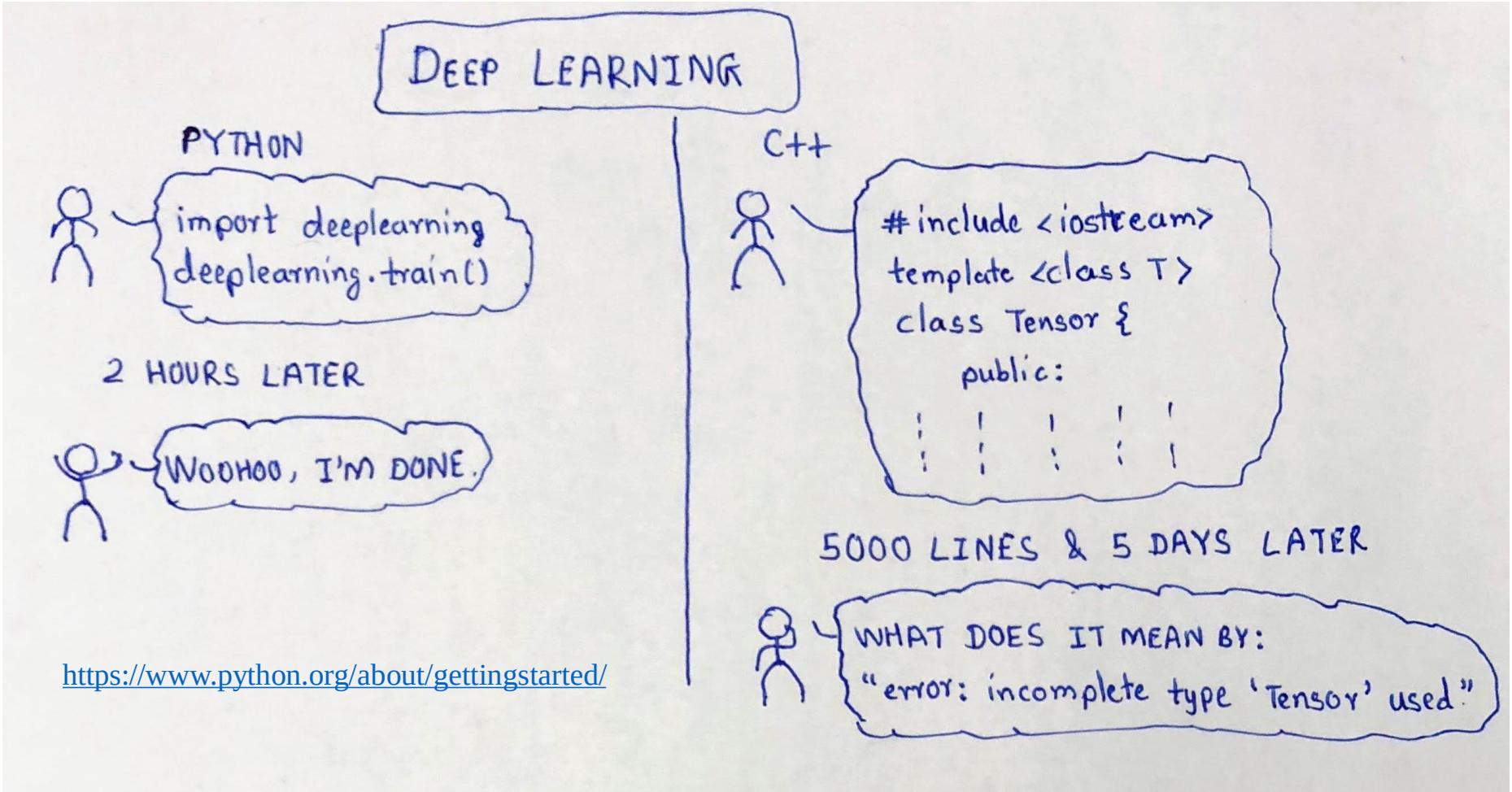
C++	Python
Advantage: • Allow interaction with low level hardware. • Allow for real time performance. • Most robotic hardware drivers are written in C/C++.	Advantage: • Easy to learn and read • Easy to access libraries • Scientific community sharing (open source, many libraries)
Disadvantage: • Learning curve • Writing and debugging programs takes a long time. • Harder to access libraries (less sharing than in Python)	Disadvantage: • Code can easily become messy for big project • Slow • Interpreted (dependencies) • Not typed (errors at runtime)

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Recommendations

Learning python first if you want to develop programs quickly and easily.

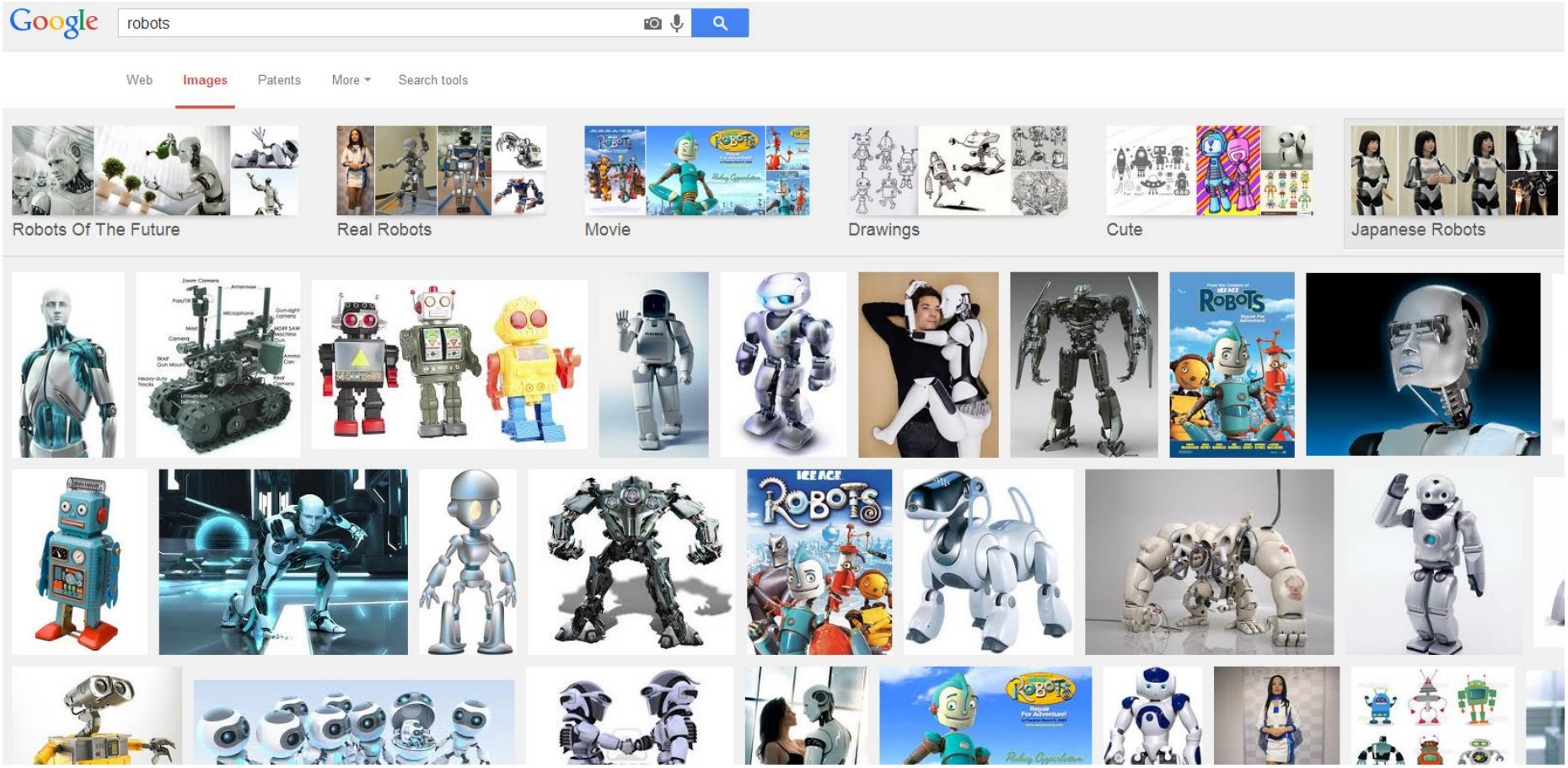


https://www.python.org/about/gettingstarted/

AncoraSIR.com

Why ROS?

Lack of standards for robotics



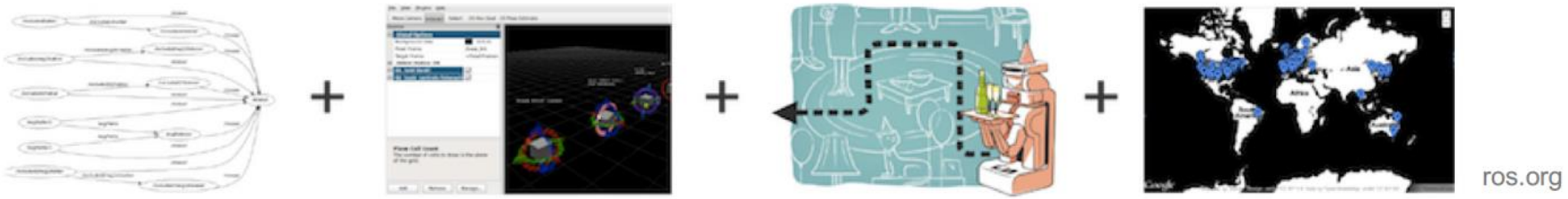
AncoraSIR.com

SUSTech
Southern University of Science and Technology

What is ROS?

a set of software libraries and tools that help you build robot applications.

ROS = Robot Operating System



Plumbing	Tools	Capabilities	Ecosystem
▪ Process management	▪ Simulation	▪ Control	▪ Package organization
▪ Inter-process communication	▪ Visualization	▪ Planning	▪ Software distribution
▪ Device drivers	▪ Graphical user interface	▪ Perception	▪ Documentation
	▪ Data logging	▪ Mapping	▪ Tutorials
		▪ Manipulation	

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage
- Since 2013 managed by OSRF (Open Source Robotics Foundation)

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Main Features

Two *"Sides"*

- The operating system side, which provides standard operating system services such as:
  - hardware abstraction
  - low-level device control
  - implementation of commonly used functionality
  - message-passing between processes
  - package management
- A suite of user contributed packages that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

ROS Philosophy

To better understand and use

- **Peer to Peer**
  - ROS systems consist of numerous small computer programs which connect to each other and continuously exchange messages
- **Tools-based**
  - There are many small, generic programs that perform tasks such as visualization, logging, plotting data streams, etc.
- **Multi-Lingual**
  - ROS software modules can be written in any language for which a client library has been written. Currently client libraries exist for C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, and more.
- **Thin**
  - The ROS conventions encourage contributors to create stand-alone libraries and then wrap those libraries so they send and receive messages to/from other ROS modules.
- **Free and open source**

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Robots Using ROS

http://wiki.ros.org/

AncoraSIR.com

ROS Installation

ROS Kinetic

- **(Recommended)** If you already have Ubuntu installed, follow the instructions at:
  - http://wiki.ros.org/kinetic/Installation/Ubuntu
- You can also download a VM with ROS Indigo Pre-installed from here:
  - http://nootrix.com/downloads/#RosVM
- Two VMs are available: one with Ubuntu 32Bits and the other with Ubuntu 64Bits (.ova files)
- You can import this file into VirtualBox or VMWare

AncoraSIR.com

Install ROS Kinetic on Ubuntu16.04

- $ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list’
- $ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
- $ sudo apt-get update
- $ sudo apt-get install ros-kinetic-desktop-full
- $ sudo rosdep init
- $ rosdep update
- $ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
- $ source ~/.bashrc

AncoraSIR.com

SUSTech
Southern University of Science and Technology

ROS Cheat Sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information.
roscreate-pkg	Creates a new ROS package.
roscreate-stack	Creates a new ROS stack.
rosdep	Installs ROS package system dependencies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rosls [package[/subdir]]
$ roscreate-pkg [package_name]
$ rosmake [package]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ rxdeps [options]
```

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

```
master
parameter server
rosout
```

Usage:

```
$ roscore
```

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in nav_msgs:

```
$ rosmsg package nav_msgs
```

List the files using sensor_msgs/CameraInfo:

```
$ rosmsg users sensor_msgs/CameraInfo
```

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:

```
$ rosrun package executable
```

Example:

Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a particular machine.
rosnode kill	Kills a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:

```
$ roslaunch -p 1234 package filename.launch
```

Launch a file in a package:

```
$ roslaunch package filename.launch
```

Launch on the local nodes:

```
$ roslaunch --local package filename.launch
```

rostopic

A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic hz	Display publishing rate of topic.
rostopic list	Print information about active topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print the name of the node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

Logging Command-line Tools

rosbag

This is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserializationof the messages.

rosbag record will generate a ".bag" file (so named for historical reasons) with the contents of all topics that you pass to it.

Examples:
Record all topics:
```
$ rosbag record -a
```
Record select topics:
```
$ rosbag record topic1 topic2
```

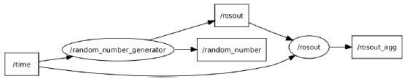
rosbag play will take the contents of one or more bag file, and play them back in a time-synchronized fashion.

Examples:
Replay all messages without waiting:
```
$ rosbag play -a demo_log.bag
```
Replay several bag files at once:
```
$ rosbag play demo1.bag demo2.bag
```

Graphical Tools

rxgraph

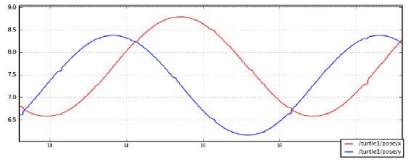
Displays a graph of the ROS nodes that are currently running, as well as the ROS topics that connect them.



Usage:
```
$ rxgraph
```

rxplot

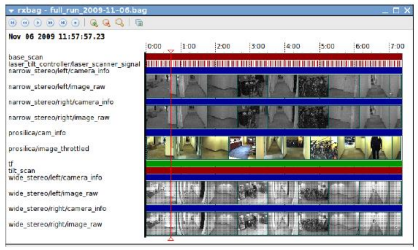
A tool for plotting data from one or more ROS topic fields using matplotlib.



Examples:
To graph the data in different plots:
```
$ rxplot /topic1/field1 /topic2/field2
```
To graph the data all on the same plot:
```
$ rxplot /topic1/field1,/topic2/field2
```
To graph multiple fields of a message:
```
$ rxplot /topic1/field1:field2:field3
```

rxbag

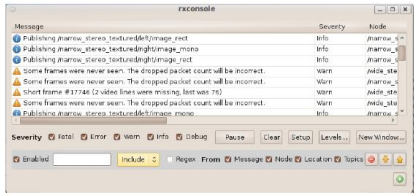
A tool for visualizing, inspecting, and replaying histories (bag files) of ROS messages.



Usage:
```
$ rxbag bag_file.bag
```

rxconsole

A tool for displaying and filtering messages published on rosout.



Usage:
```
$ rxconsole
```

tf Command-line Tools

tf_echo

A tool that prints the information about a particular transformation between a source_frame and a target_frame.

Usage:
```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

Examples:
To echo the transform between /map and /odom:
```
$ rosrun tf tf_echo /map /odom
```

view_frames

A tool for visualizing the full tree of coordinate transforms.

Usage:
```
$ rosrun tf view_frames
$ evince frames.pdf
```


ROS Wiki

- http://wiki.ros.org/

Installation

- http://wiki.ros.org/ROS/Installation

Tutorials

- http://wiki.ros.org/ROS/Tutorials

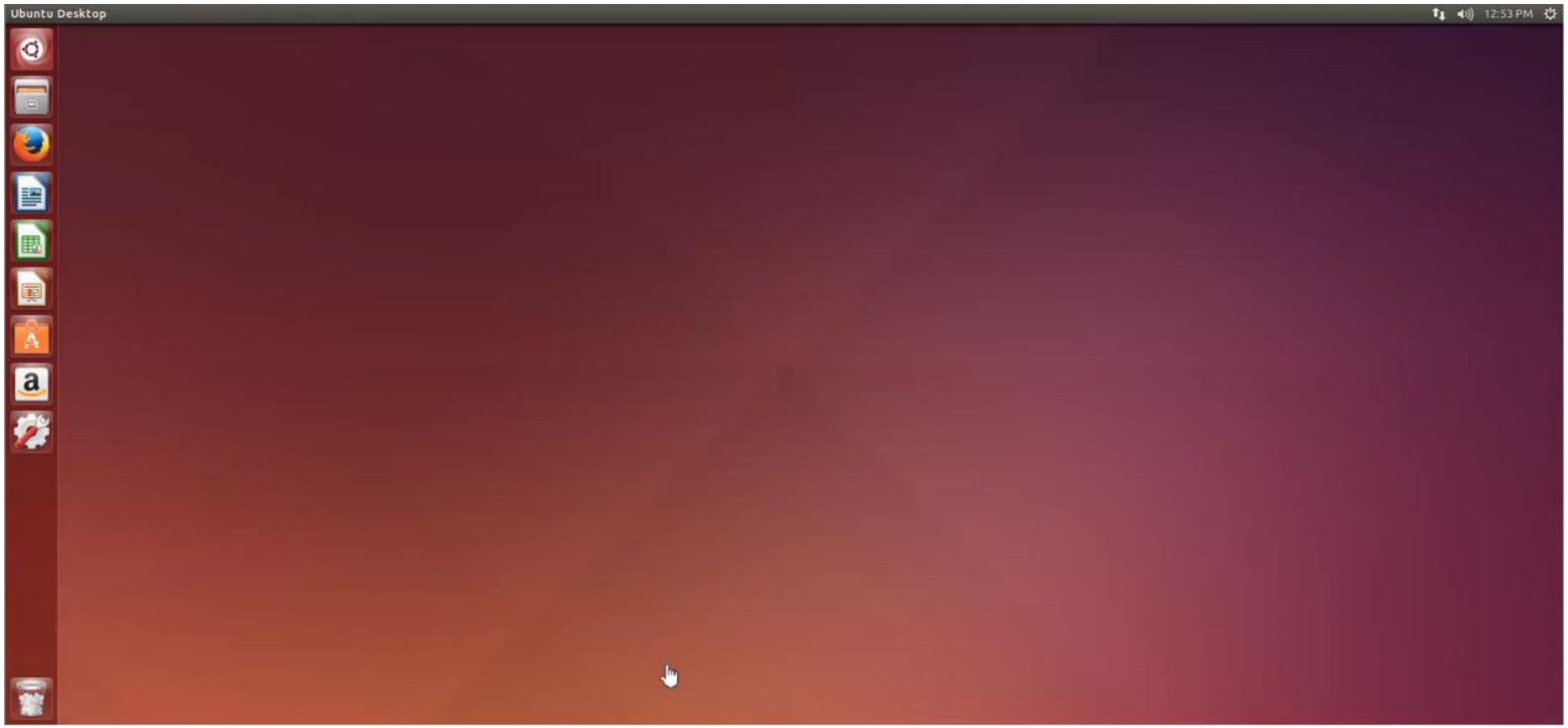
ROS Cheat Sheet

- http://www.tedusar.eu/files/summerschool2013/ROScheatsheet.pdf

Run Your First Example

Turtlesim Example in ROS

- Start ROS Master — $ roscore
- Start turtle bot — $ rosrun turtlesim turtlesim_node
- Start operating node and control the turtle bot with your keyboard — $ rosrun turtlesim turtle_teleop_key



AncoraSIR.com

Thank you!

Prof. Song Chaoyang

- Dr. Wan Fang (sophie.fwan@hotmail.com)

AncoraSIR.com