Robotics Lab: Setup your PC

Week 1

Mario Selvaggio

This is a description of how to set up and move your first steps in a native Linux Ubuntu and ROS installation. Additionally, instructions on how to install and get started with Git version control systems and Docker containers are provided.

Linux

Linux is family of open-source Unix-like operating systems based on the Linux kernel (first release 1991). Ubuntu is one of the most popular Linux distributions, it is released every six months, with Long-Term Support (LTS) releases every two years. New releases make the system compatible with new hardware. The last LTS release at writing time is 24.04 LTS. To get maximum support it generally good to choose the second-most recent LTS distribution.
Why do we use Linux? It is safe, it can be easily configured and customized for your needs, it is fast.

Installation

Download and install the desktop image which is appropriate for your machine. For this course it is recommended to install Ubuntu 22.04 LTS (Jammy Jellyfish). You will need at least 15 GB of space in your root Ubuntu partition to install and work with ROS (recommended 40-50 GB). Comprehensive installation instructions can be found here. After installation, open a terminal and update your installation to the newest version with

```
$ sudo apt update
$ sudo apt upgrade
```

If you are not familiar with how to use the Linux command line, have a look at this tutorial. Some basic Linux commands are given in the following section.

Basic Linux commands

In the following, fundamental commands you will be using along the course are explained. To test a command it must be typed into a terminal as indicated. We recommend you to use terminator, that allows you to have multiple terminals in one window. It can be installed with

```
$ sudo apt update
$ sudo apt install terminator
```

`man`: is used to display the user manual of any command that we can run on the terminal It provides a detailed view of the command which includes name, synopsis, description, options, and other information. Example use

```
$ man ls
```

`pwd`: print working directory. When you first open the terminal, you are in the home directory of your user. Use this command to know which directory you are currently in. It gives us the absolute path, which means the path that starts from the root

```
$ pwd
```

`ls`: lists the files of the current directory. You can see hidden files using the option `-a`, e.g.

```
$ ls -a
```

`cd`: changes the directory. It takes as argument the destination folder path. When you are in the `home` folder, and you want to go to the `Downloads` folder, you can use

```
$ cd Downloads
```

To navigate to the upper-level directory

```
$ cd ..
```

To navigate to the `home` directory

```
$ cd
```

`mkdir` & `rmdir`: used to create or remove a folder

```
$ mkdir newFolder
$ rmdir newFolder
```

`touch`: used to create a file. For instance

```
$ touch newFile.txt
```

`rm`: used to delete a file. Using the option `-r` deletes recursively all the elements inside a directory

```
$ rm -r
```

`cp`: used to copy files. It takes two arguments as follows

```
$ cp src_file dest_file
```

`mv`: used to move (rename) files. It takes two arguments as follows

```
$ mv text new
```

`locate`: used to find a file in a Linux system

```
$ locate file
```

Remember to to locate newly created files

```
$ sudo updatedb
```

`echo`: used to add data to a text file

```
$ echo "hello, my name is Mario" > newFile.txt
```

`cat`: displays the content of a file

```
$ cat /home/$USER/.bashrc
```

`sudo`: command with administrative or root privileges

```
$ sudo nano /etc/hosts
```

`chmod`: used to make a file executable and to change the permissions

```
$ chmod +x numbers.py
```

when your application needs to access to USB devices

```
$ chmod 777 /det/ttyUSB0
```

`ping`: to check your connection to a server

```
$ ping www.google.it
```

`grep`: print lines matching a pattern. If you want to search the occurrence of a word into a text file

```
$ grep -i "string" file
```

the recursive option `-R` to search the occurrence in multiple file
|: (pipe) redirects the output of a command (left side) to another command

```
$ cmd1 | cmd2
$ ls | grep "string"
```

Robot Operating System (ROS 2)

Installation

Akin to Linux distributions (e.g. Ubuntu), a ROS distribution is a versioned set of ROS packages. For the this course, we recommend installing ROS 2 Humble.
You have 2 options:

1. Installing it on the native Ubuntu installation following these instructions

2. Use the ROS2 Docker container (see later) which is provided to you by the instructor.

Important: For the installation of ROS you have to configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse". Please follow the Ubuntu guide for instructions on how to do this. Log out, then log in again.

Check your installation

Open a Terminal window and run the following commands

```
$ source /opt/ros/foxy/setup.bash
  $ ros2 run demo_nodes_cpp talker
```

In another terminal, run a `turtlesim` node

```
$ source /opt/ros/foxy/setup.bash
  $ ros2 run demo_nodes_py listener
```

You should see the talker publishing messages and the listener saying it is hearing those messages. This verifies both the C++ and Python APIs are working properly.

Setup your workspace

Once you have a working ROS installation. Open a terminal and source the environment with

```
$ source /opt/ros/humble/setup.bash
```

If you do not want to do this for every terminal you open, run the following command

```
$ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
```

this adds the source command to your `.bashrc` file, that is sourced every time you open a new shell (terminal). Create a workspace wit the following command

```
$ mkdir -p ~/ros2_ws/src
$ cd ~/ros2_ws
```

Build the workspace

```
$ colcon build
```

Source your workspace with

```
$ source install/local_setup.bash
```

Git - Version Control System (VCS)

A VCS is used to track modifications to a source code repository. It tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. A detailed historical record of the projects life allows to instantly revert the codebase back to a previous point in time. By far, the most widely used modern version control system in the world today is Git. To install Git on your machine follow the following instructions

```
$ sudo apt update
$ sudo apt install git
```

Check your installation

```
$ git --version
```

Configure your Git username and email

```
$ git config --global user.name "Emma Paris"
$ git config --global user.email "eparis@atlassian.com"
```

Working with Git

A repository is a git-tracked folder, commits are used to create snapshots of your folder content. Branches are history of commits that can be merged at some point. To setup a local repository

```
$ cd /path/to/your/existing/code
$ git init
```

If a project has already been set up in a remote repository

```
$ git clone <repo url> <folder name>
```

If you use git clone to set up your local repository, it is already configured for remote collaboration. If you used `git␣init` to make a fresh repo, you'll have no remote repo to push changes to. You can configure it by

```
$ git remote add origin <remote_repo_url>
```

Once you have linked the remote repo you can `push` local branches to it

```
$ git push -u origin <local_branch_name>
```

The git `add` command adds a change in the working directory to the staging area, while the git `commit` command captures a snapshot of the project's currently staged changes

```
$ git add <files>
$ git commit -m "commit message"
```

Example

```
$ cd /path/to/project
$ echo "test content for git tutorial" >> CommitTest.txt
$ git add CommitTest.txt
$ git commit -m "added CommitTest.txt to the repo"
```

The git `status` command displays the state of the working directory and the staging area

```
$ git status
```

The git `log` command displays committed snapshots

```
$ git log
```

When you have found a commit reference to the point in history you want to visit, you can utilize the git checkout. Checking out a specific commit will put the repo in a "detached HEAD" state. This means you are no longer working on any branch. From the detached HEAD state, we can execute

```
$ git checkout -b new_branch_without_crazy_commit
```

This will create a new branch and switch to that. At this point, can continue work on this new branch. `git revert` is the best tool for undoing shared public changes, git `reset` is best used for undoing local private changes

```
$ git revert HEAD
```

will create a new commit with the inverse of the last commit

```
$ git reset --hard a1e8fb5
```

In this way, commits no longer exist in the commit history but if we have a shared remote repository git will assume that the branch being pushed is not up to date.
The `git pull` command is used to fetch and download content from a remote repository and immediately update the local repository to match that content

```
$ git pull <remote>
```

You might find useful working with a GUI

```
$ sudo apt-get install git-gui
```

Git and ROS

...

Github

You can use www.github.com to setup your remote repo or cloning an existing one. After you sign up, you have to generate a personal access token (password) in Settings → Developer Settings → Personal access token (classic) and click on generate new (classic) token.

Docker

Docker is tool for managing virtualization entities in the OS. Using docker is useful when you want to optimize the development, testing, and deployment of your robotic application. Your software will be coming with its dependencies and libraries in an entity called **container**. A container are the live, running instances of docker **images** that contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs. Using docker allows spending less time installing the correct versions of libraries and software or understanding what is wrong with the installed libraries. Docker is less resource-intensive than virtual machines that emulate hardware and hosts a whole operating system. You can find more information about docker here.

Install Docker

To install docker on your Linux Ubuntu you need to run the following commands:

```
$ sudo apt update
$ sudo apt install apt-transport-https curl gnupg-agent ca-certificates software-
  properties-common -y
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
  focal stable"
$ sudo apt install docker-ce
```

Working with Docker

Now you can use docker commands, let's try installing a simple image

```
$ sudo docker run hello-world
```

Note that the `hello␣world` image does not exist locally, and is pulled from docker hub. **Note:** Superuser permissions are requested. If you want to avoid invoking `sudo` you can add the currently logged-in user to the docker group

```
$ sudo usermod -aG docker $USER
$ newgrp docker
```

Check the images in your stack with

```
$ docker images
```

To remove images from your stack

```
$ docker image rmi <image_id> -f
```

You can create and start a container from a given image. For instance, the following code

```
$ docker create -it --name ubu1 ubuntu /bin/bash
$ docker start -i ubu1
```

creates the container `ubu1` out of the `ubuntu` image.
You can use `docker run` to simultaneously create and start a new container. The syntax is the following

```
$ docker run -it --name <container_name> <image_name> /bin/bash
```

To check what are the containers running, in a terminal you can type

```
$ docker ps
```

Use the option `-a` if you want to list all the containers, regardless of their state.
Restart, stop or removing a container is done by

```
$ docker restart <container_name>
$ docker stop <container_name>
$ docker rm <container_name>
```

To save an image to a file

```
$ sudo docker save img_name -o <saved_img_name>
```

To load an image from a file

```
$ sudo docker load -i <img_file_name>
```

If you work with a versioned image, you might want to do this after you have modified and committed the container with

```
$ docker commit c3f279d17e0a svendowideit/testimage:version3
```

Note: Files created into containers belong to the container user, cannot be modified from outside, and are lost if you remove the container. It is good practice to create and share a folder in your computer as a volume. To do this, you can add `-v` option to `docker run` command, for example

```
docker run -v </local_folder>:<container_folder>:rw ... --name=<container_name> <
image> bash
```

To let the container share the host network you can use the `--network` option. You have options to share video, and so on. As you add options, typing the command into a terminal can be difficult. In ubuntu you can create `.sh` script files containing a sequence of commands.
An example is provided below

```
#!/bin/bash

xhost +

docker run -it --privileged -v /dev/bus/usb:/dev/bus/usb \
--env=LOCAL_USER_ID="$(id -u)" \
-v ~/dev:home/dev/:rw \
-v /tmp/.X11-unix:tmp/.X11-unix:ro \
-e DYSPLAY=:0 \
--network host \
--workdir="/home/dev/" \
--name=ros1-noetic osfr/ros:noetic-desktop bash
```

You can find all the docker scripts useful for this course at this link https://github.com/RoboticsLab2024/ros2_docker_scripts.

C++ Programming

ROS2 is language agnostic: you can use either C++ or Python to develop your robotic application. C++ is fast and versatile, it can be used both for high-level reasoning and for low-level control, especially if you are chasing performance. Python is a high-level programming language, very useful for sensor elaboration, learning and similar. It can be used if you don't need performance.
We will be using C++ during the course. It is recommended to refresh your C++ skills using any C++ tutorial, e.g. https://www.learncpp.com/. Basic concepts behind C++ programming are provided in the following.

Create your first program

In a folder, create a file `hello_world.cpp` and paste the following code inside

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

In a terminal, navigate to your working folder and execute the following command to compile

```
$ cc hello_world.cpp -o hello_world
```

or

```
$ gcc hello_world.cpp -o hello_world
```

If it does not work try

```
$ sudo apt install build-essential
```

To execute your program, run it with

```
$ ./ hello_world
```

If you see some printed output, you have successfully created your first C++ program.

Classes

A class represents user-defined data types grouping together related pieces of information. Example: Robot class.

```
#include <string>
#include <iostream>

using namespace std;

// Create a Robot class with some attributes and methods
class Robot {
  public:
    Robot(string _n, int _x, int _y)
    {
      robot_name = _n;
      positionX = _x;
```

```
      positionY = _y;
    }
    string getName(){return robot_name;}
    int getPositionX(){return positionX;};
    int getPositionY(){return positionY;};
    void move(int _x, int _y){positionX = _x; positionY = _y;} ;

  private:
    string robot_name = "";
    int positionX;
    int positionY;
};

int main() {
  // Define variables
  int position_x = 1;
  int position_y = 2;
  string name = "my_robot";

  // Create an object of the Robot class
  Robot r(name, position_x, position_y);
  cout << r.getName() << " is created in x = " << r.getPositionX()
  << ", y = " << r.getPositionY() << " position \n";

  // Use the move() method of the Robot class
  r.move(3,4);
  cout << r.getName() << " is moved in x = " << r.getPositionX()
  << ", y = " << r.getPositionY() << " position \n";

  return 0;
}
```

Pointers

Allow the data manipulation in a flexible way. Manipulating the memory addresses of data can be more efficient than manipulating the data itself. In C++ `&x` evaluates the address of the variable `x` in memory, `*(&x)` takes the address of `x` and dereferences it. An example program `pointers.cpp` is provided below

```
#include <iostream>

using namespace std;

int main() {

    int traj_length = 6;
    // assign the trajectory as integer array
    int robot_trajectory[traj_length] = {1,2,3,4,5,6};

    cout << "The initial trajectory is:" << endl;
    for(int i = 0; i < traj_length; i++){cout << robot_trajectory[i] << endl;}
```

```
    cout << endl;

    // declare a pointer to an integer array
    int* robot_trajectory_ptr;
    // assign
    robot_trajectory_ptr = &robot_trajectory[0];
    // modify the trajectory acting on its pointer
    robot_trajectory_ptr[2] = 10;

    cout << "The modified trajectory is:" << endl;
    for(int i = 0; i < traj_length; i++){cout << robot_trajectory[i] << endl;}
    cout << endl;

    return 0;
}
```

Make & CMake

Make is a building tool that automates building process and is typically used when you have a complex compilation structure for your program. To compile using `make` create a makefile in your src folder containing

```
all: pointers

pointers: pointers.o
  g++ -o pointers pointers.cpp
```

and compile your program running

```
  $ ./ make
```

CMake automatizes the generation of the makefile. It acts in two stages

```
  $ ./ cmake
```

generates makefile using configuration file `CMakeLists.txt`. After you can compile using

```
  $ ./ make
```

A minimal example file is

```
# CMakeLists files in this project can
# refer to the root source directory of the project as ${POINTERS_SOURCE_DIR} and
# to the root binary directory of the project as ${POINTERS_BINARY_DIR}.
cmake_minimum_required (VERSION 2.8.11)
project (POINTERS)

include_directories("${CMAKE_CURRENT_SOURCE_DIR}")

# Add executable called "Pointers" that is built from the source files
# "pointers.cxx". The extensions are automatically found.
add_executable (Pointers pointers.cpp)
```