AAE3004
Dynamical Systems and Control

Lab 1a: Autonomous ROS car control project

Instructors: Dr Weisong Wen, Mr Jian LIU (TA), and Mr Baoshan Song (TA)
Assistant Professor
Department of Aeronautical and Aviation Engineering
The Hong Kong Polytechnic University
Office: PQ408
Tel: 3400 8234
Email: welson.wen@polyu.edu.hk

AAE3004

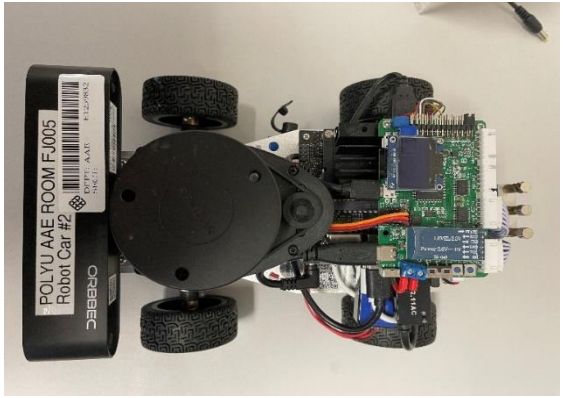
- 1. Robot Car Platform Overview
- 2. Linux Operating System
- 3. ROS

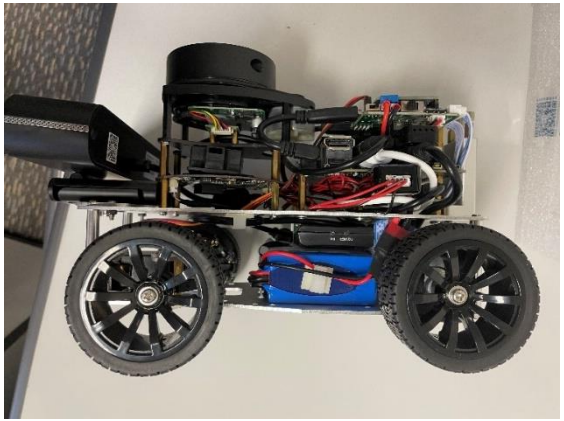
AAE3004

Part I – Overview of ROS Car Platform

1. Robot Car Platform Overview

- raspberry pie and STM32 motherboards
- Supporting ros2 / opencv
- Multiline lidar
- Visual processing
- deep learning framework (Yolo/Tensorflow)
- navigation and map building





Part II – Linux Operating System

2. Linux Operating System (OS)

What is Operating system?

- windows system for PC,
- IOS system for Apple products,
- Android system for mobile devices

2. Linux Operating System (OS)

What is Operating system?

Operating System is a ***program*** that mediates between the ***user*** and the computer ***hardware***.

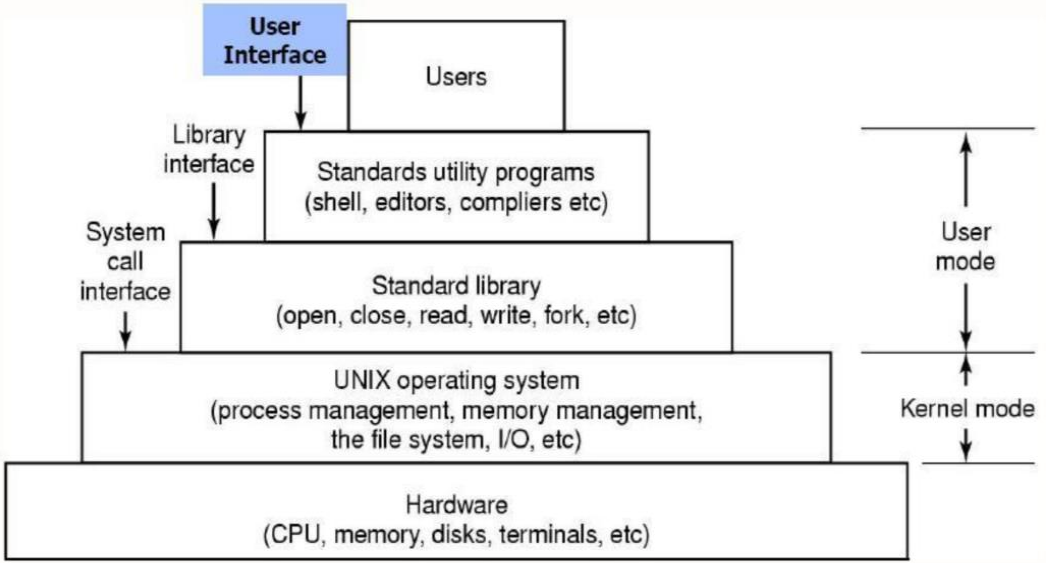
• Hides hardware details.

• Manages software resources
– resources means objects necessary to execute the program, e.g. memory, processor (CPU), input/output, communication ports
– strategies for allocation and deallocation of resources

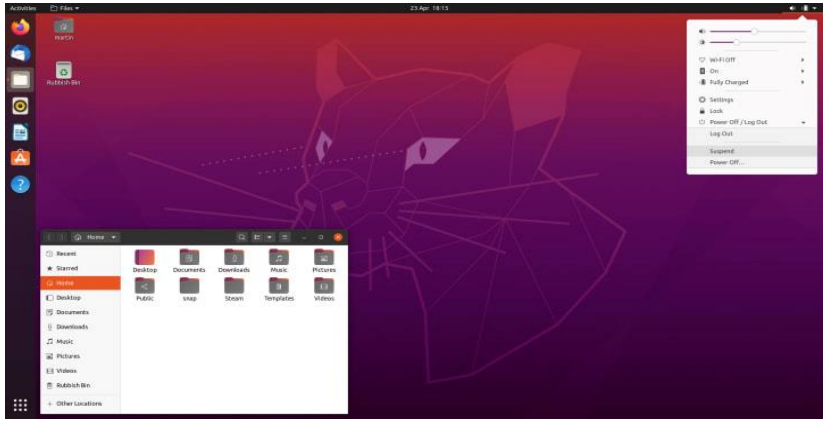
• Other activities: security, job accounting, error detecting tools, etc.

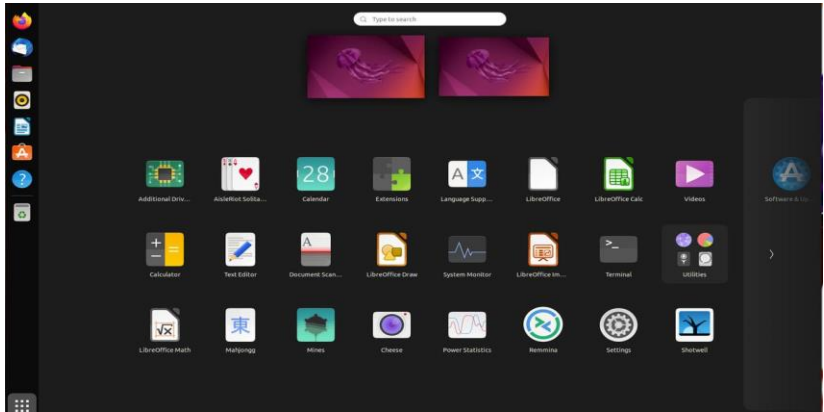
2. Linux Operating System

Linux Structure



Graphic User Interface (GUI)





2. Linux Operating System

Linux History

Linus Torvalds, Finland, born in the same year as UNIX, i.e. 1969, creator of the Linux kernel and the Git version control sysem.

Linus Torvalds announcing Linux 1.0, 30.03.1994

Linus Torvalds in 2019

Richard Stallman, founder of the GNU project and the Free Software Foundation, co-creator of the GNU GPL license, creator of the Emacs editor, GCC compiler, GDB debugger.

Richard Stallman in 2019

May 1991, version 0.01: no support for the network, limited number of device drivers, one file system (Minix), processes with protected address spaces
The Linux Kernel Archives – https://www.kernel.org/
– **2022-02-23**, latest stable version **5.16.11**
– **2022-02-20**, latest mainline **5.17-rc5**

2. Linux Operating System

Linux Basic Feature

- **Multi-access** system (with time sharing) and multi-tasking
- **Multi-process** system, simple mechanisms to create hierarchy of processes, kernel preemption
- **Available for many architectures**
- Simple standard user interface that can be easily replaced (shell, command interpreter)
- File systems with a multi-level tree structure
- Shared libraries, loaded into memory dynamically (one code used simultaneously by many processes)
- Compliance with the POSIX 1003.1 standard
- Different formats of executable files

Free Distribution!

2. Linux Operating System

Tutorial

1. The Linux Programming Interface: A Linux and UNIX System Programming

The **Linux Programming Interface** offers **in-depth information** about the system and library functionalities.

2. How Linux Works: What Every Superuser Should Know

How Linux Works is a conceptual book that explains brief information about the **Linux internals**. This book also explains how Linux firewalls, interfaces, and networks work.

3. Linux Tutorial for Beginners: Introduction to Linux Operating System

https://www.youtube.com/watch?v=V1y-mbWM3B8

4. Linux Command Line Tutorial For Beginners

https://www.youtube.com/watch?v=YHFzr-akOas&list=PLS1QulWo1RIb9WVQGJ_vh-RQusbZgO_As

5. Ubuntu Complete Beginners Guide (Full Course in one video!)

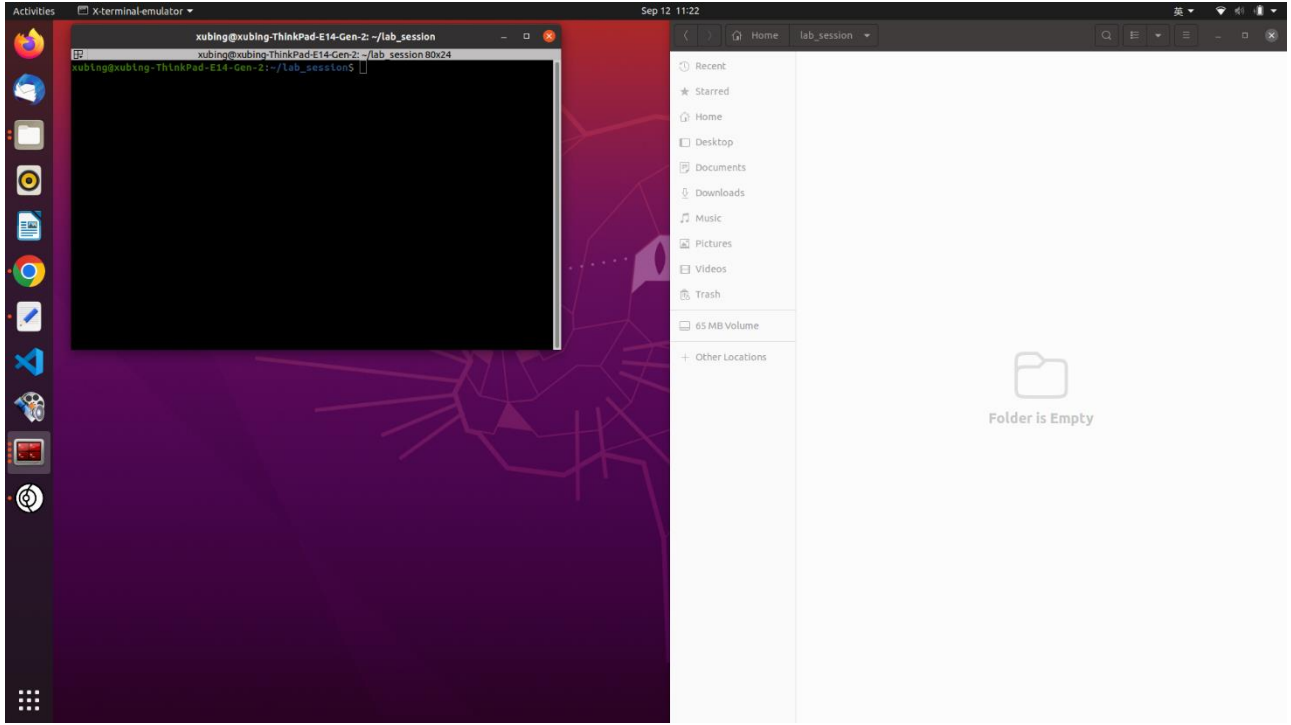
https://www.youtube.com/watch?v=D4WyNjt_hbQ&t=36s

2. Linux Operating System

Helloworld demo with Linux

Step 1: Open the terminal

- **right click on the margin**
- **Ctrl+Alt+T**

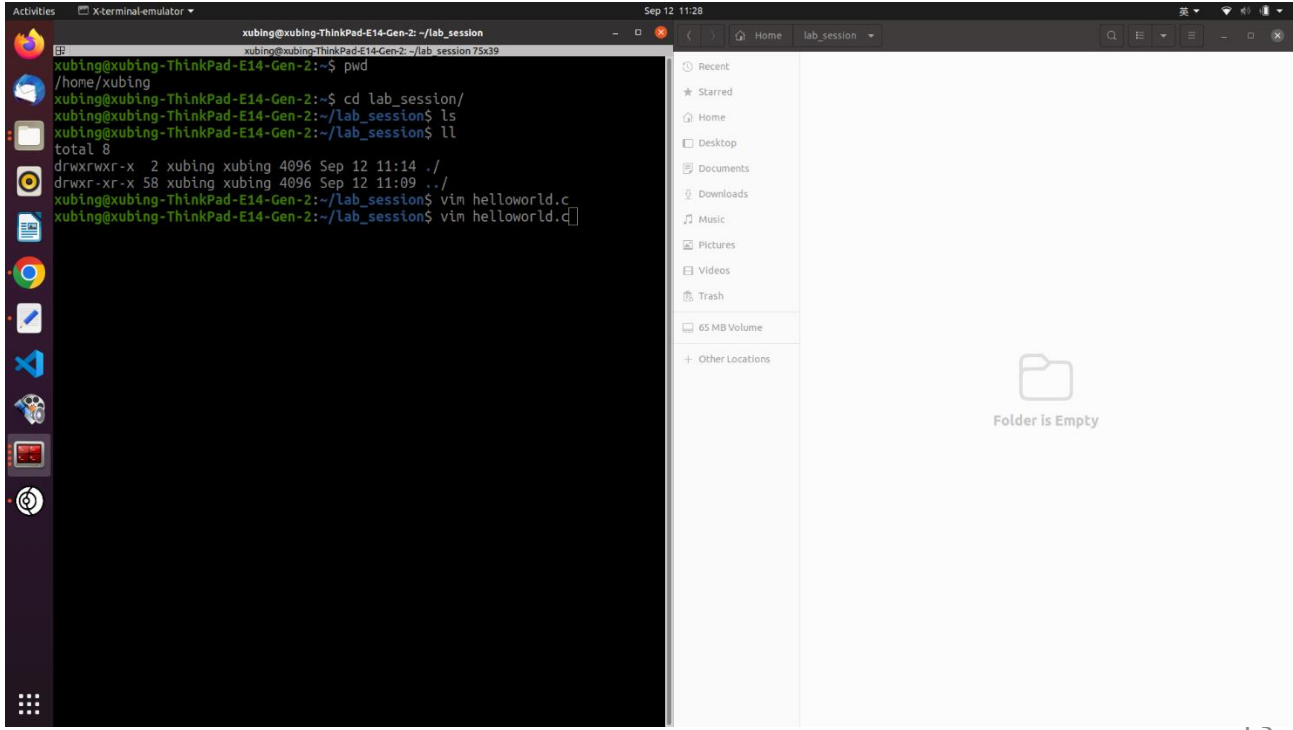


2. Linux Operating System

Helloworld demo with Linux

Step 2: Create the C file

- $ vim helloworld.c

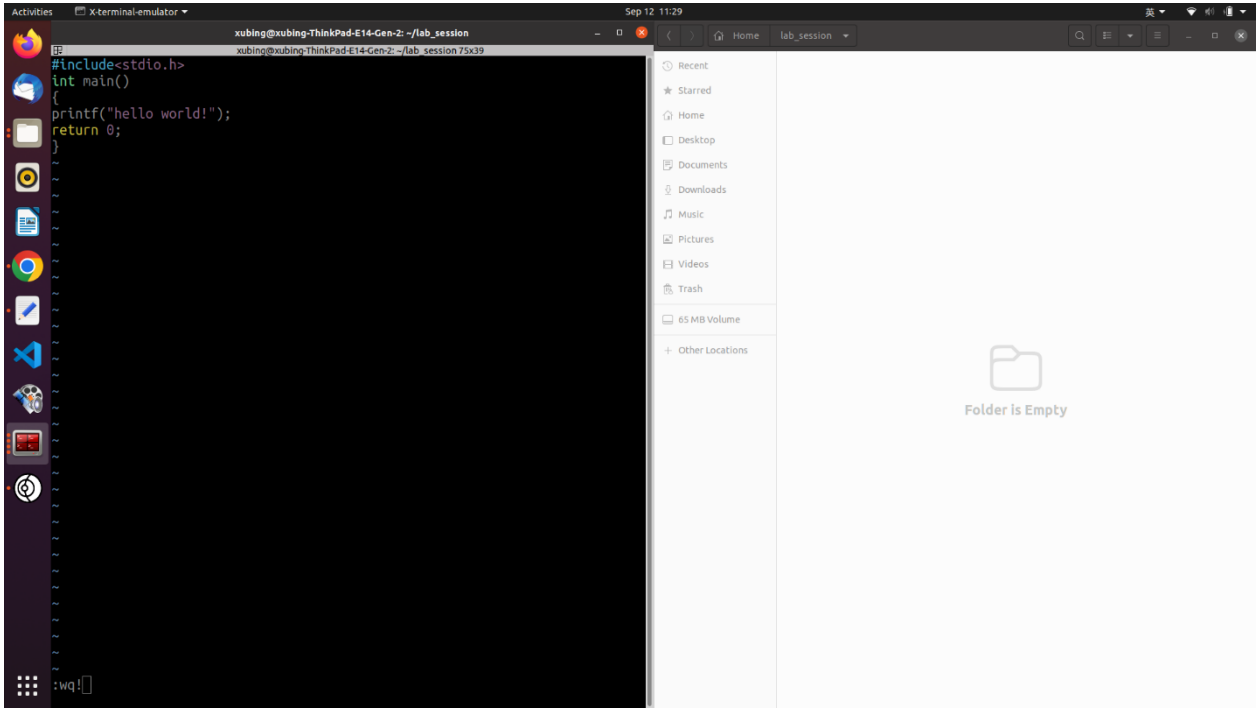


2. Linux Operating System

Helloworld demo with Linux

Step 3: Input the code and save it

```
#include<stdio.h>
Int main()
{
printf(“hello world!”);
return 0;
}
```

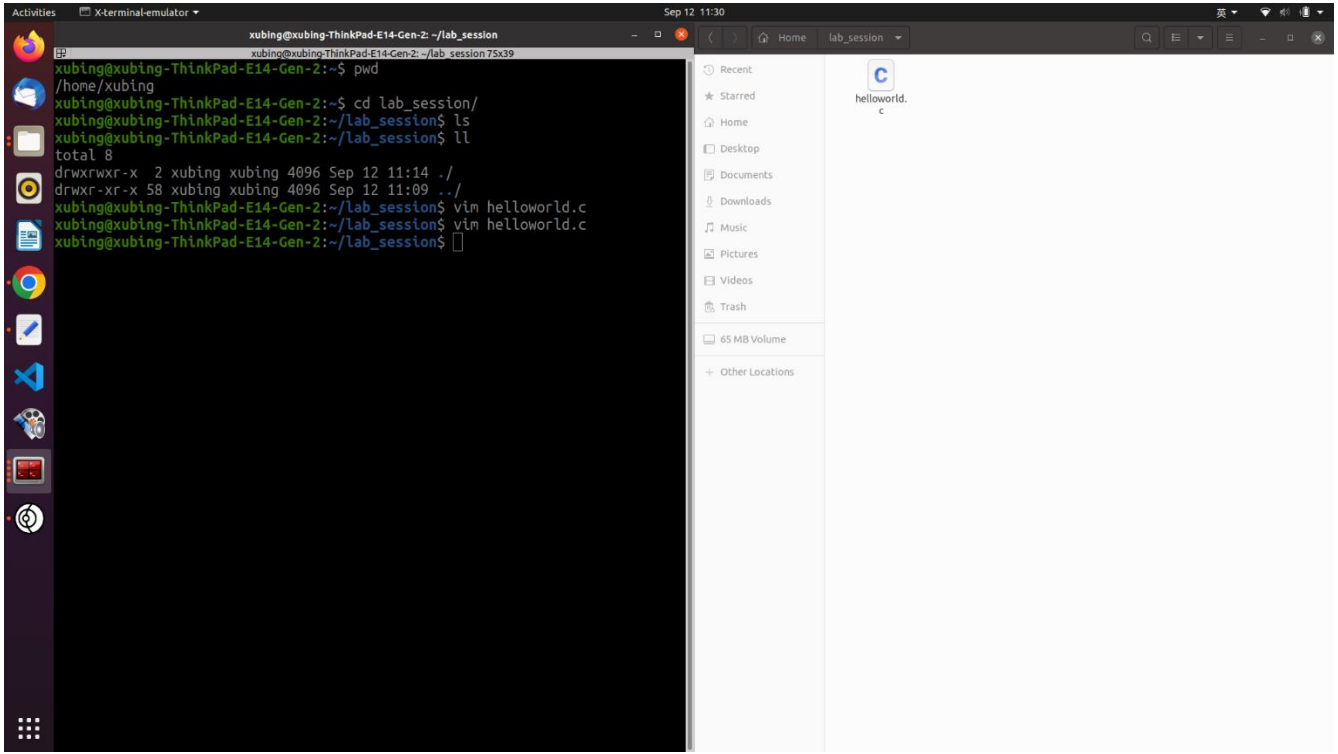


2. Linux Operating System

Helloworld demo with Linux

Step 3: Input the code and save it

```
#include<stdio.h>
Int main()
{
printf(“hello world!”);
return 0;
}
```

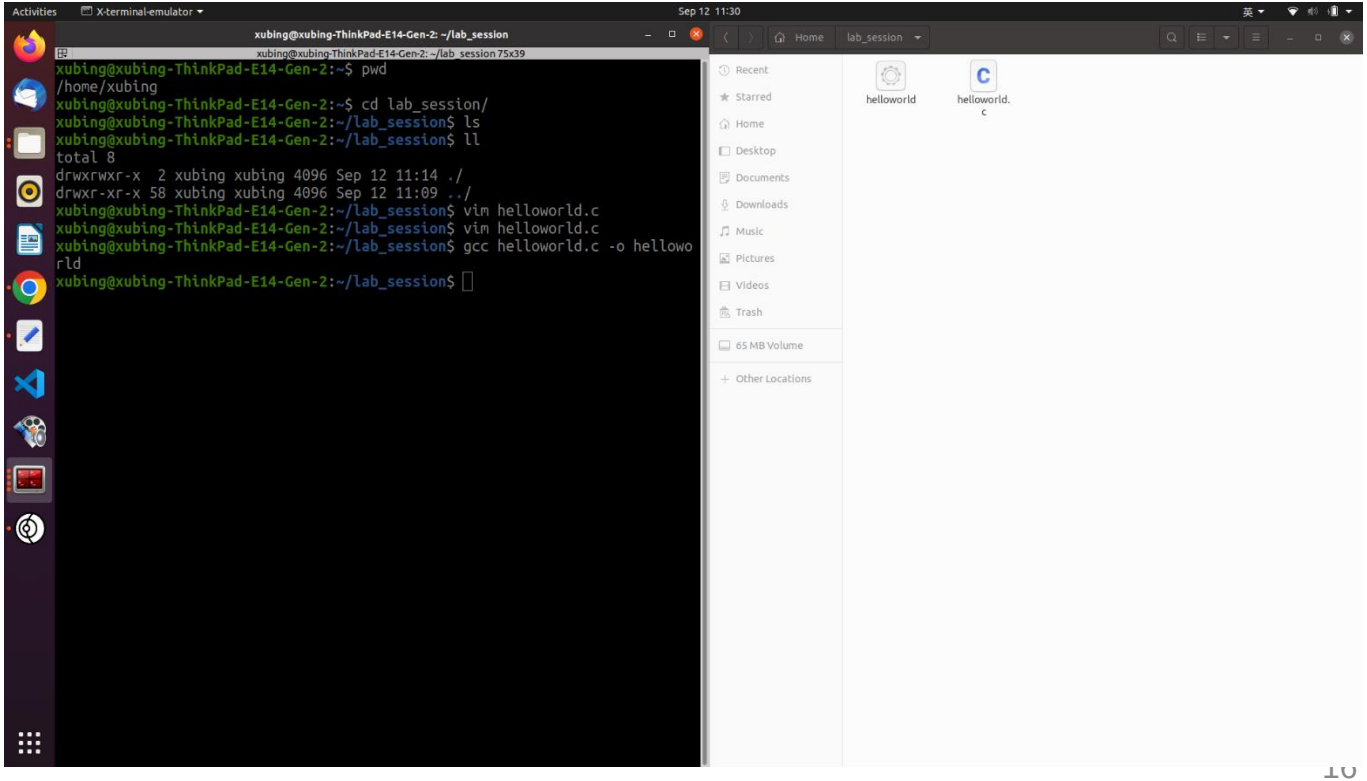


2. Linux Operating System

Helloworld demo with Linux

Step 4: Compile

- **$ gcc helloworld.c –o helloworld**

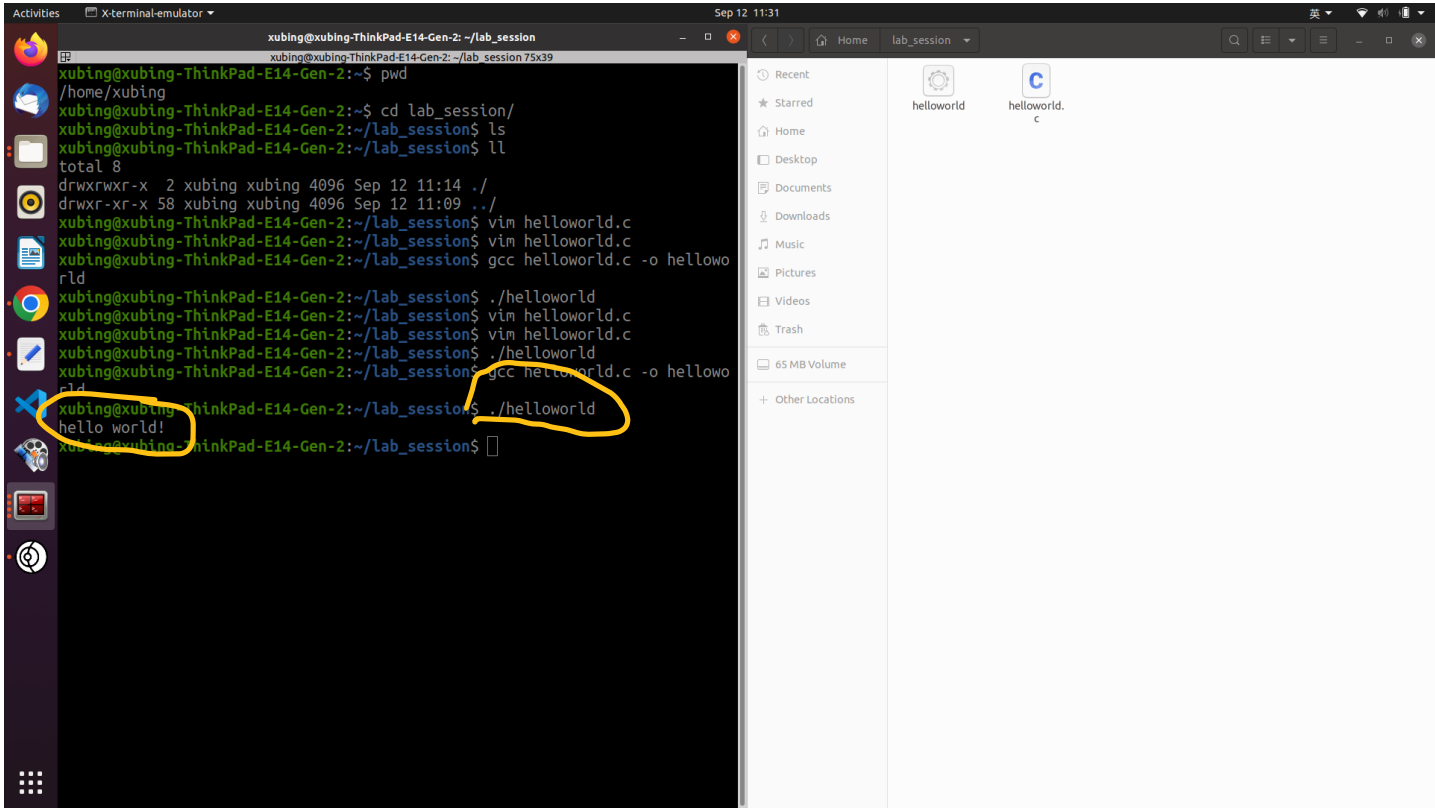


2. Linux Operating System

Helloworld demo with Linux

Step 5: Execute

- ./helloworld

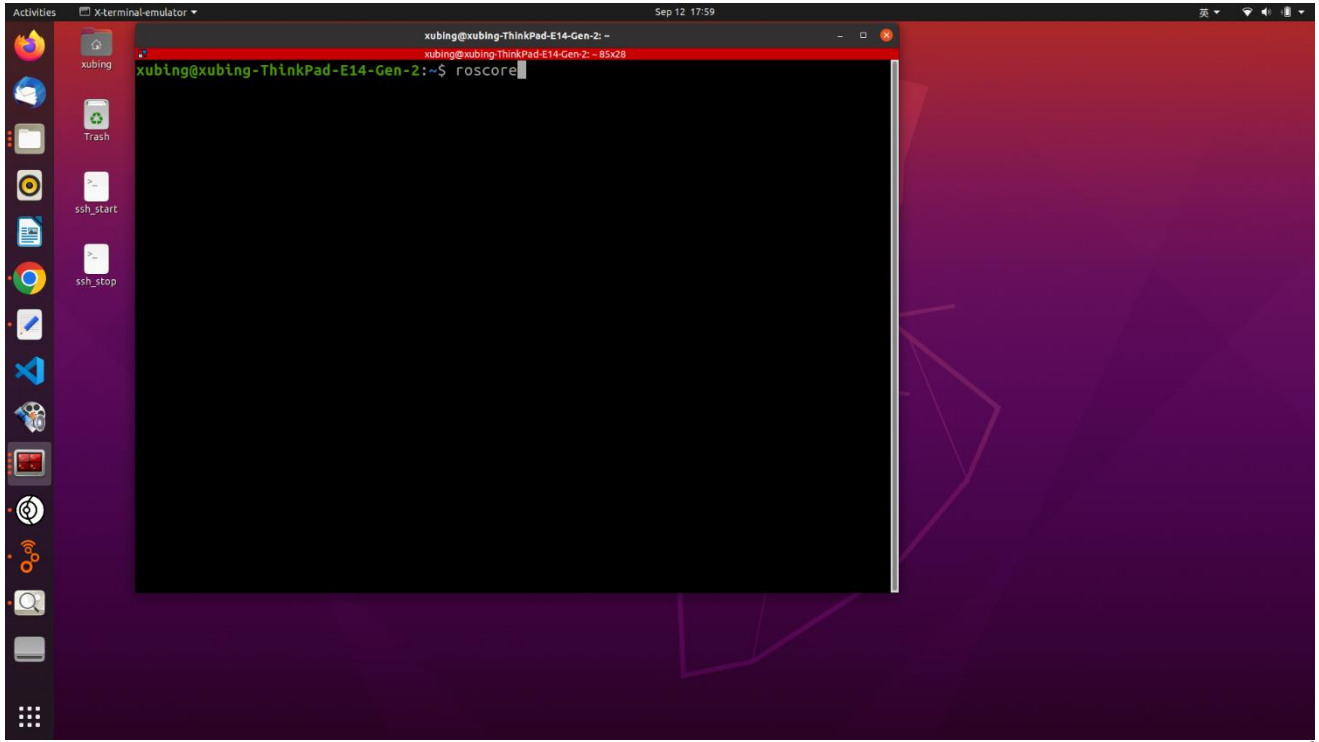


2. Linux Operating System

ROS turtles demo

Step 1: create the master node

- **$ roscore**

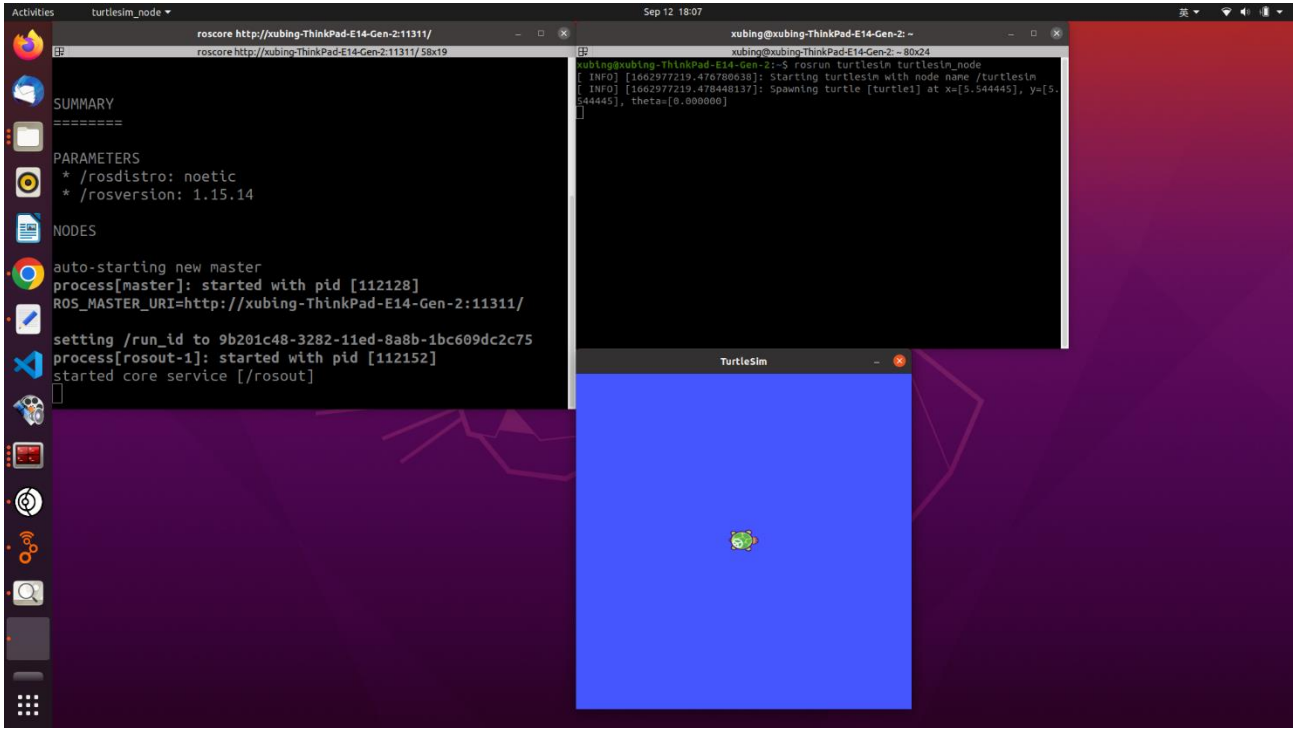


2. Linux Operating System

ROS turtles demo

Step 2: create a node to show turtle

- **$ rosrun turtlesim turtlesim_node**

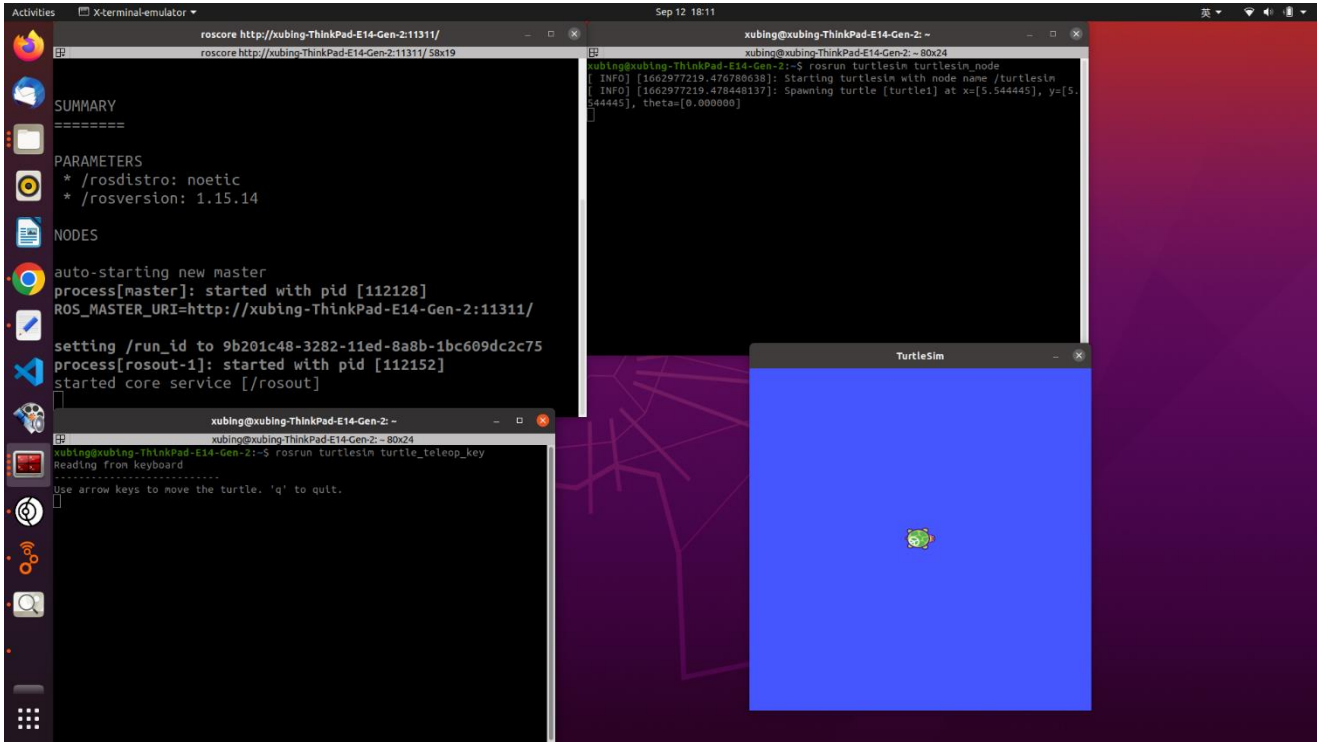


2. Linux Operating System

ROS turtles demo

Step 3: create the keyboard node

- **$ rosrun turtlesim turtlesim_node**

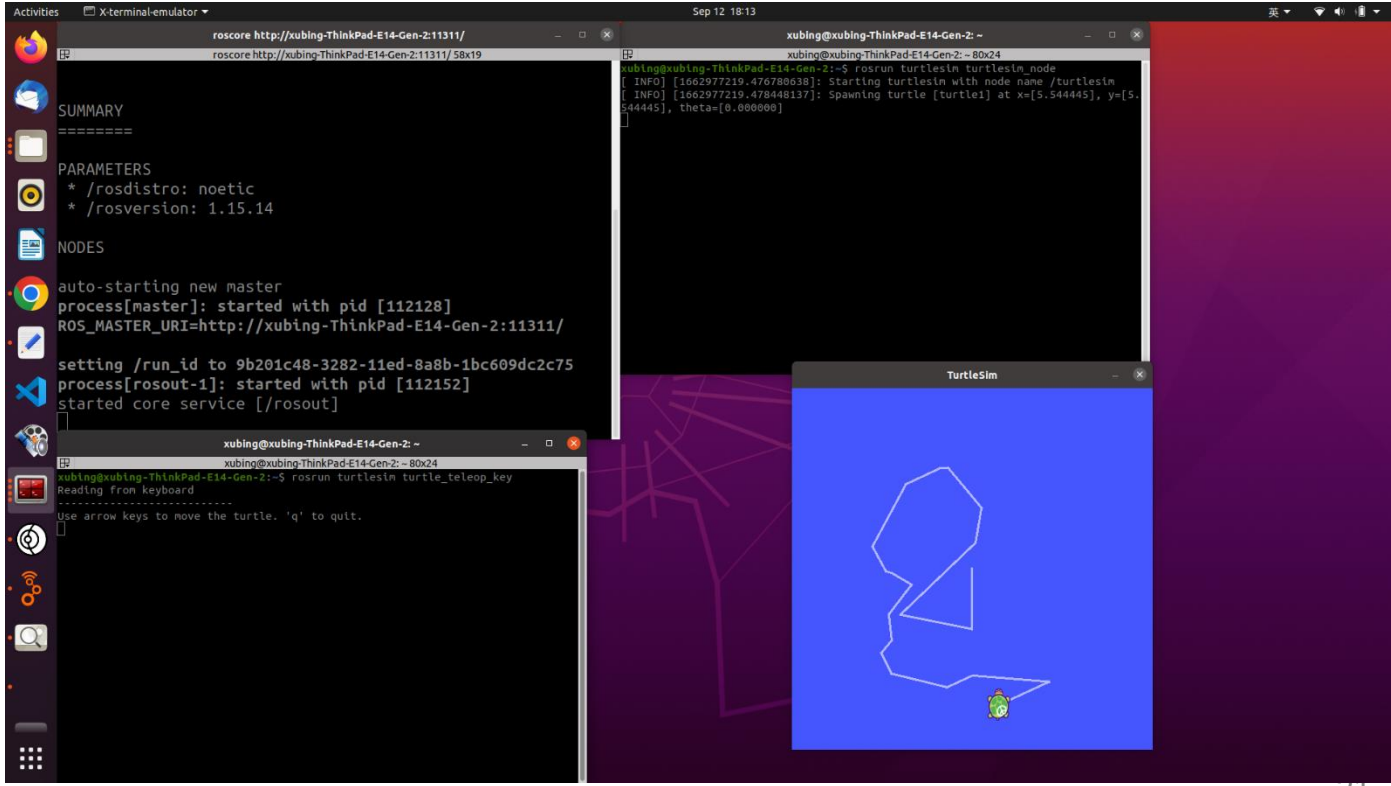


2. Linux Operating System

ROS turtles demo

Step 4: control the turtle

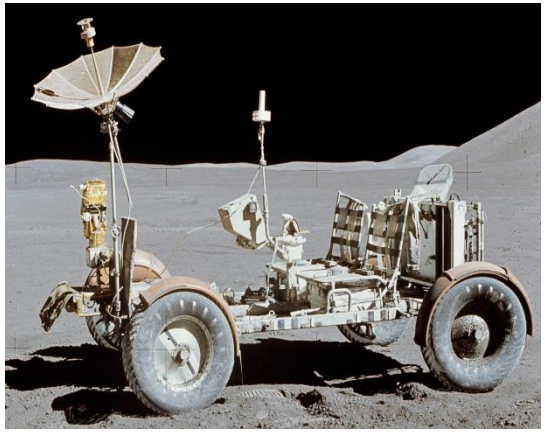
↑ ↓ ← →



Part III – ROS

3. ROS— Robot Operating System

Mechanical Arm



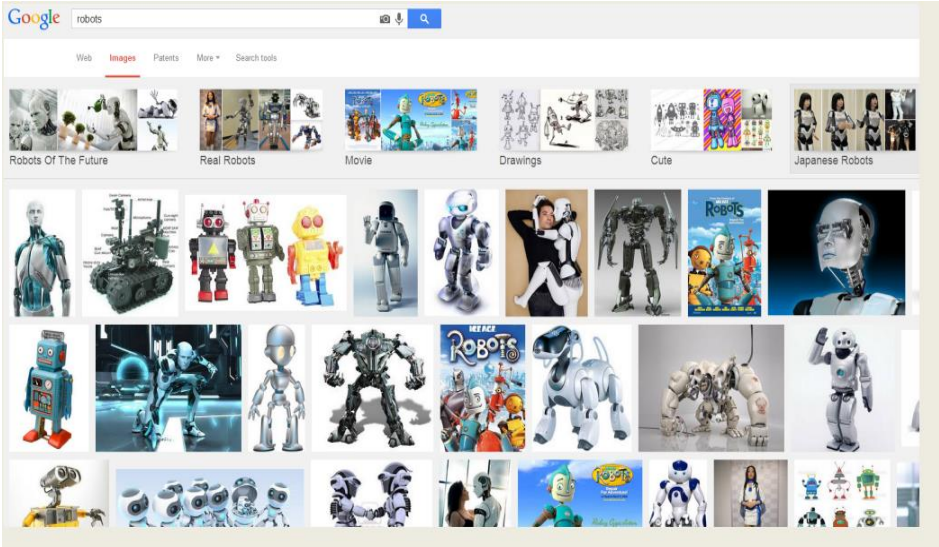
Lunar Rover

Humanoid Robot

The Problem:
Lack of standards for robotics

3. ROS— Robot Operating System

- ROS is an **open-source** robot operating system
- A set of software libraries and tools that help you build robot applications that work across a wide variety of robotic platforms
- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory and development continued at Willow Garage

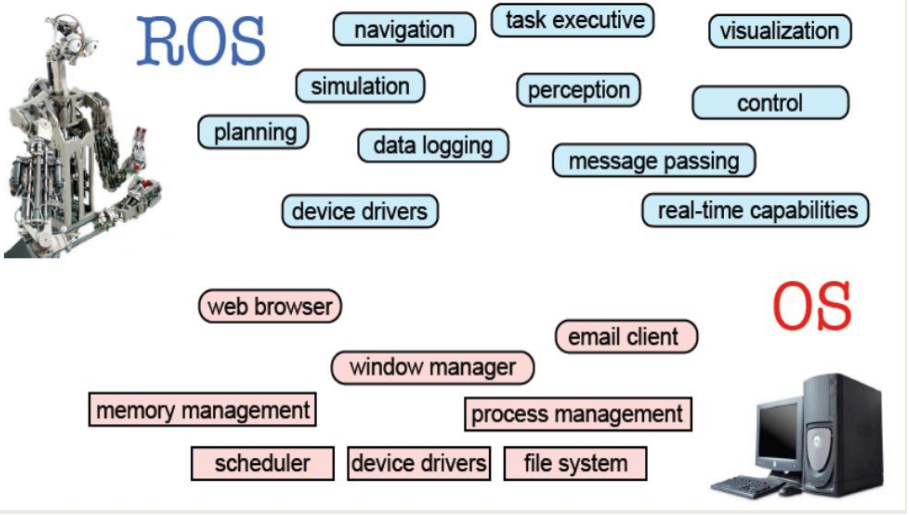


- Since 2013 managed by OSRF (Open Source Robotics Foundation)

3. ROS

ROS Main Features

- 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

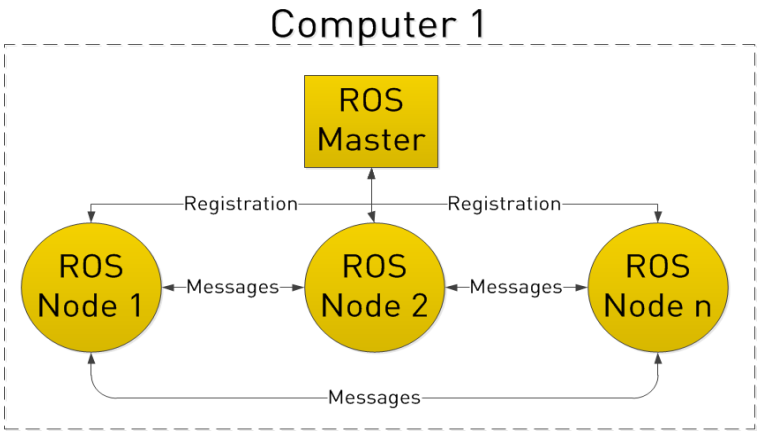


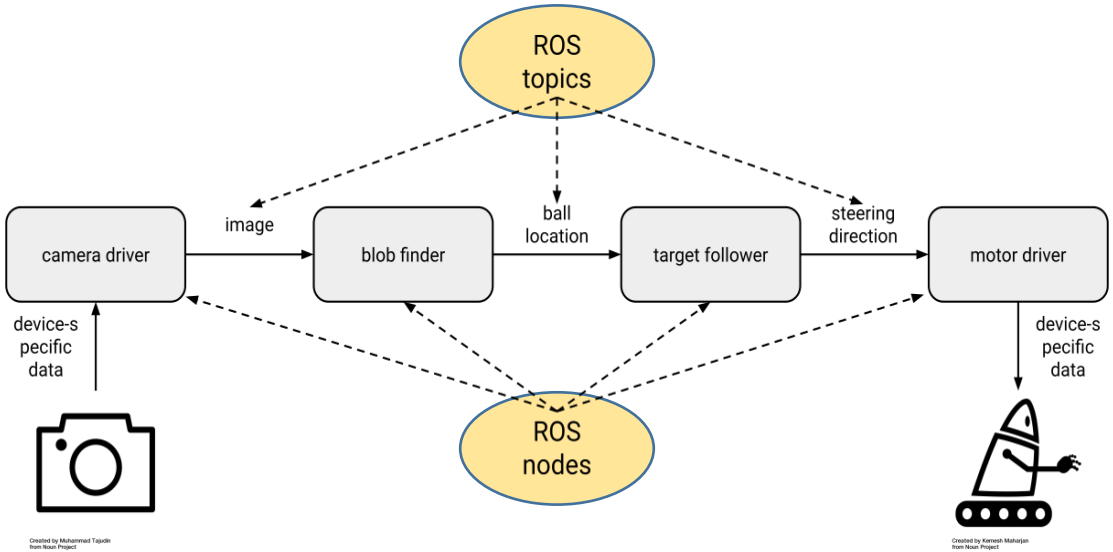
- A suite of user contributed packages that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.

3. ROS

ROS Computation graph model

The Computation Graph is the peer-to-peer **network of ROS processes** that are processing data together.





Decentralized !!

3. ROS

ROS Core Concepts

- ***Nodes***

(1)Single-purposed executable ***programs*** – e.g. sensor driver(s), actuator driver(s), mapper, planner, UI, etc.

(2) Individually compiled, executed, and managed

(3) Nodes can **publish** or **subscribe** to a Topic

(4) Nodes can also provide or use a service

(5) Nodes are written using a ROS client library

– roscpp – C++ client library

– rospy – python client library.

3. ROS

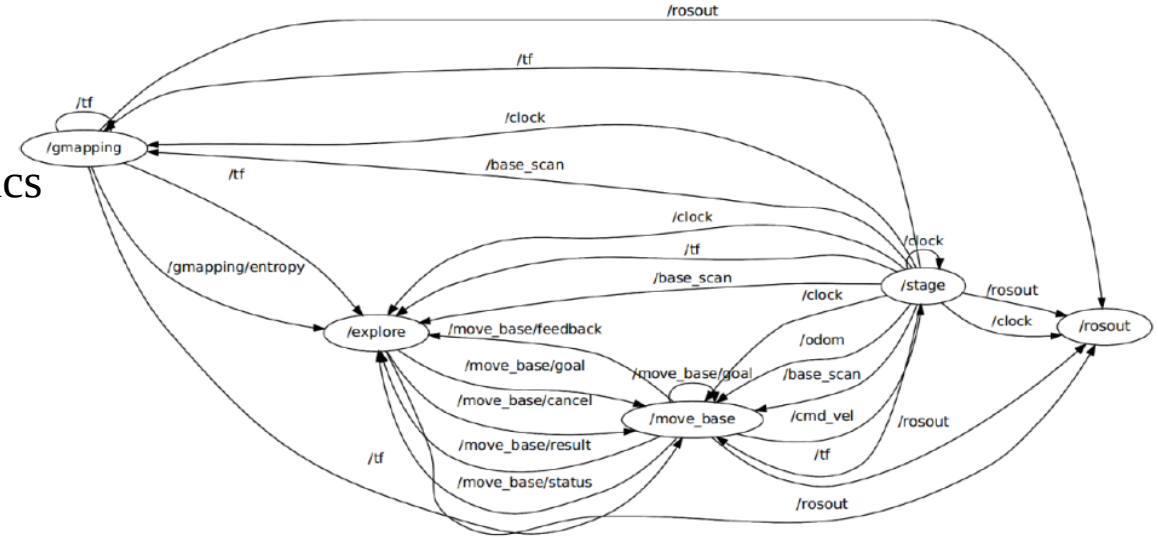
ROS Core Concepts

- ***Topics*:** Topics are named buses over which nodes exchange messages.

– e.g., data from a laser range-finder might be sent on a topic called scan, with a message type of LaserScan

(2) Nodes communicate with each other by publishing messages to topics

(3) Publish/Subscribe model: 1-to-N broadcasting



ROS Topics Graph

3. ROS

ROS Core Concepts

- ***Messages*:** Strictly-typed data structures for inter-node communication

For example, geometry_msgs/Twist is used to express velocity commands:

```
Vector3 linear
Vector3 angular
```

Vector3 is another message type composed of:

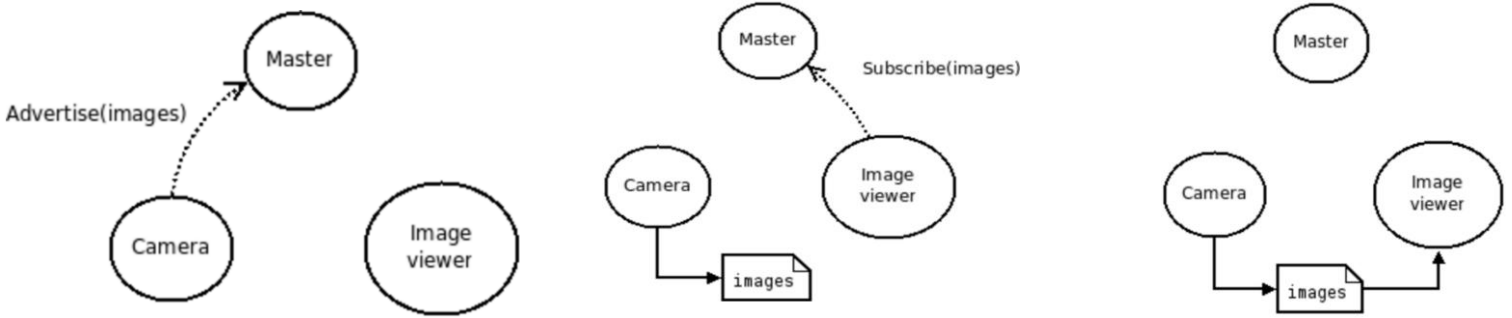
```
float64 x
float64 y
float64 z
```

3. ROS

ROS Core Concepts

- ***ROS Master***

Provides connection information to nodes so that they can transmit messages to each other
– Every node connects to a master at startup to register details of the message streams they publish, and the streams to which that they to subscribe.
– When a new node appears, the master provides it with the information that it needs to form a direct peer-to-peer connection with other nodes publishing and subscribing to the same message topics.



3. ROS

ROS Core Concepts

- ***ROS Services*:** Synchronous inter-node transactions

(1)Service/Client model: 1-to-1 request-response

(2)Service roles:
- carry out remote computation
- trigger functionality / behavior

(3) Example:
- map_server/static_map – retrieves the current grid map used by the robot for navigation

3. ROS

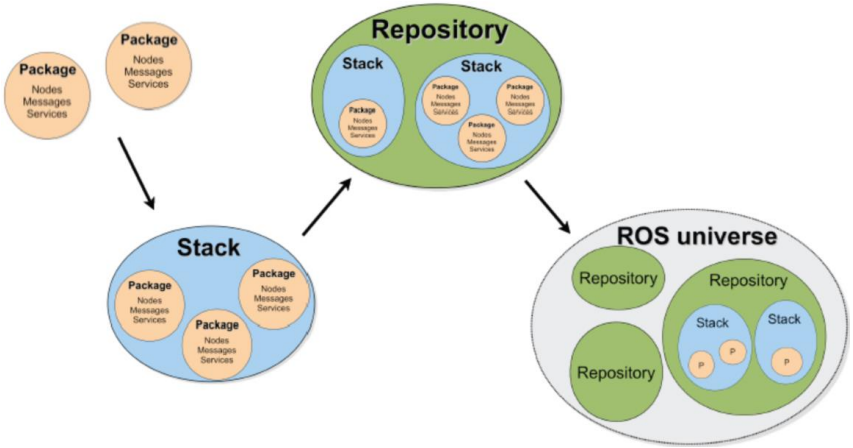
ROS Core Concepts

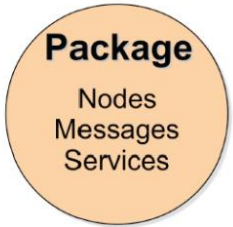
- ***ROS Package***

(1)Software in ROS is organized in packages.

(2) A package contains one or more nodes and provides a ROS interface.

(3) Most of ROS packages are hosted in GitHub.





3. ROS

Tutorial

- http://wiki.ros.org/

Installation:

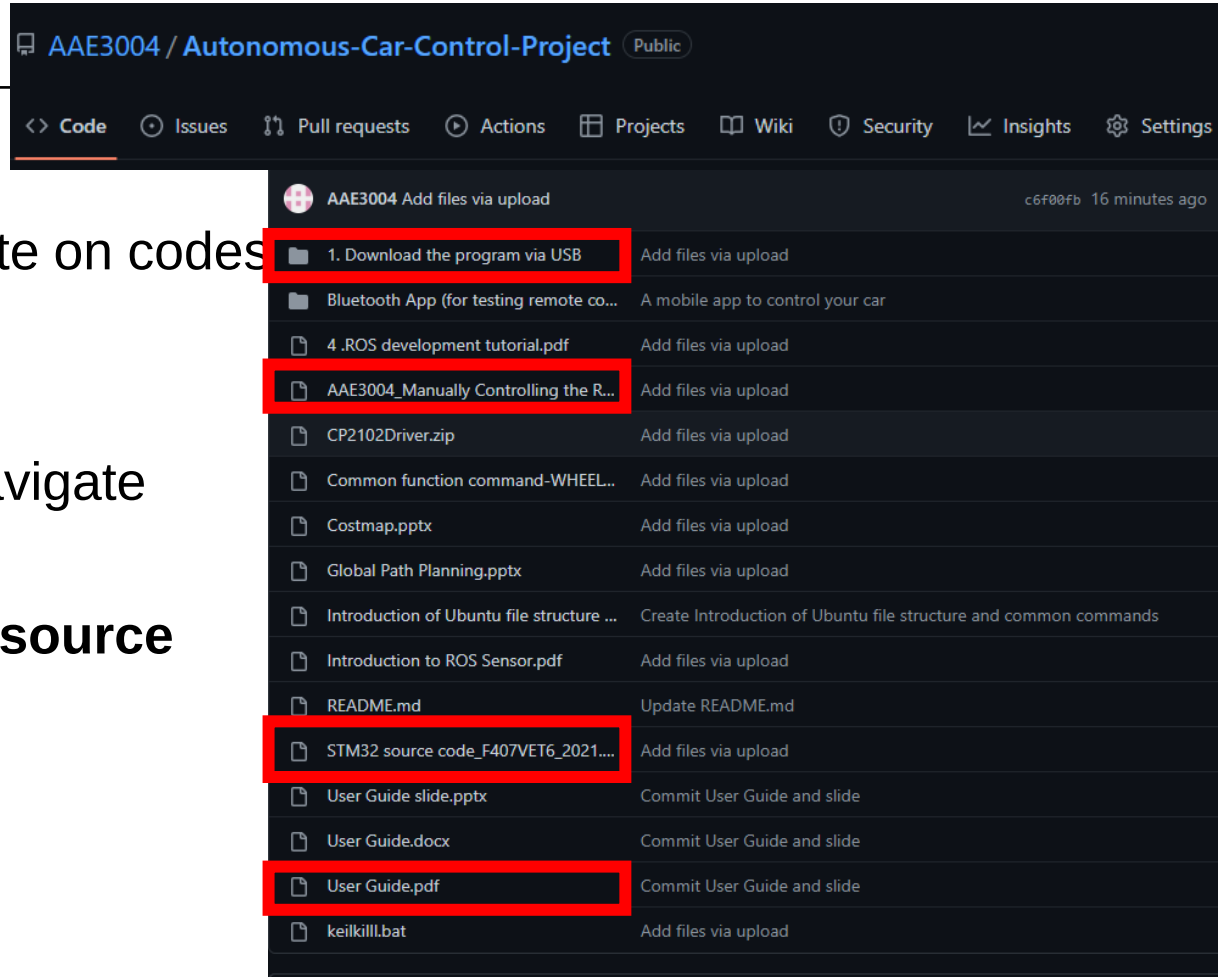
- http://wiki.ros.org/ROS/ Installation

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

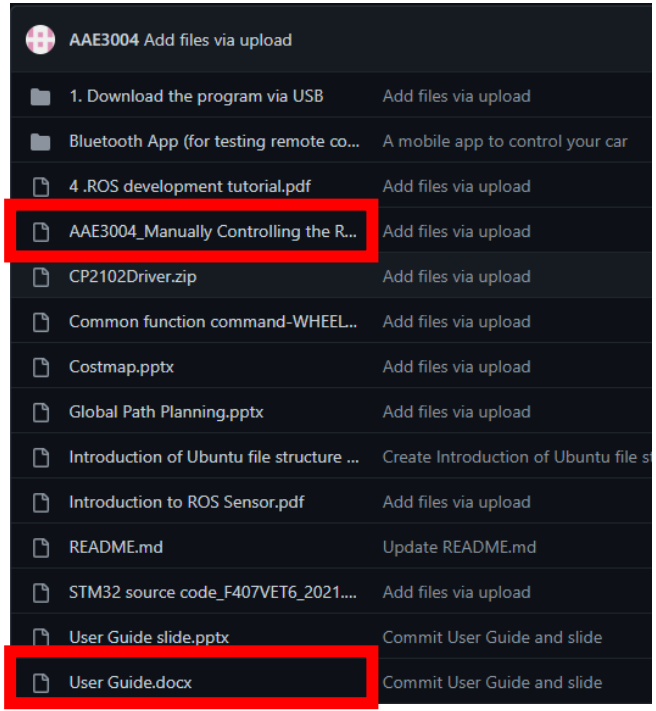
- ROS Tutorial Videos
http://www.youtube.com/playlist? list=PLDC89965A56E6A8D6
- ROS Cheat Sheet
http://www.tedusar.eu/files/summerschool2013/ ROScheatsheet.pdf

GitHub

Online platform to collaborate on codes
– Issues and Pull requests
Controlling the car
Using LiDAR to map and navigate
STM32 source code
Tutorials to install STM32 source code



Controlling the car



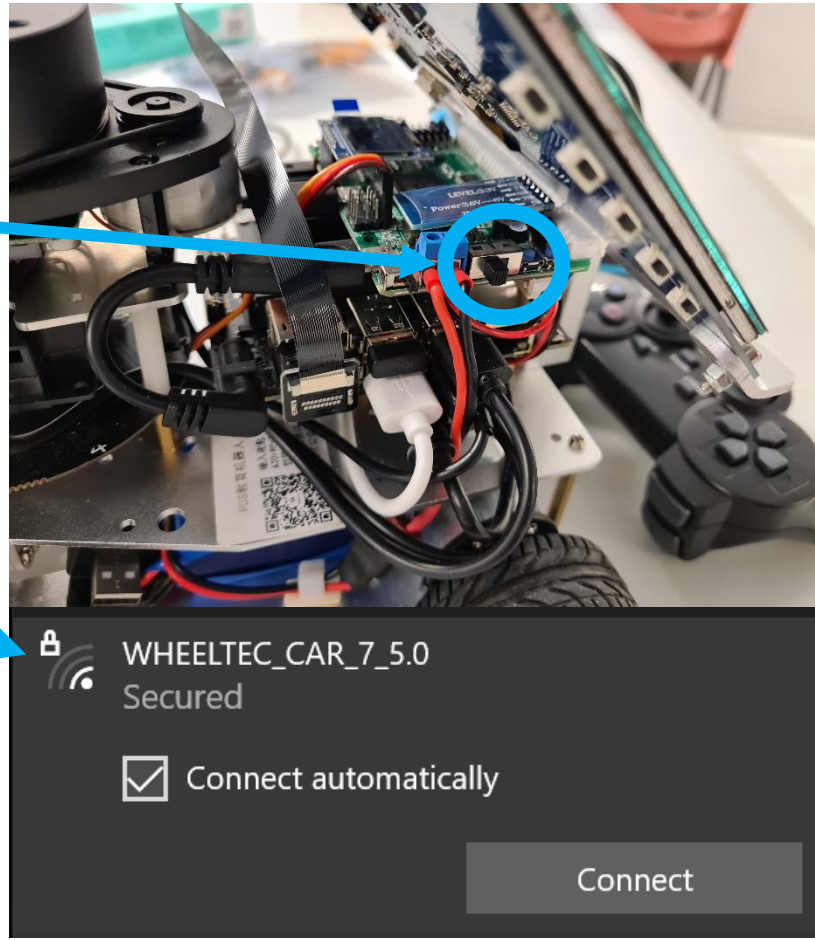
1. Using PS2 controller
2. Manually on the car - *AAE3004_manually Controlling the ROS Car*
3. Remotely using VM – *User Guide*

https://github.com/AAE3004/Autonomous-Car-Control-Project/tree/main

Keyboard control - Steps

1. Turn on the ROS car
2. Install Ubuntu on a Virtual Machine
3. Connect the Ubuntu to the ROS car Wi-Fi

Password: dongguan



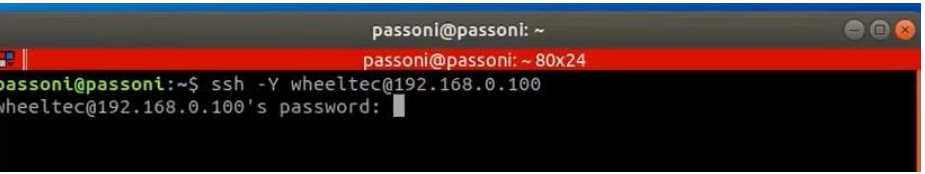
Steps

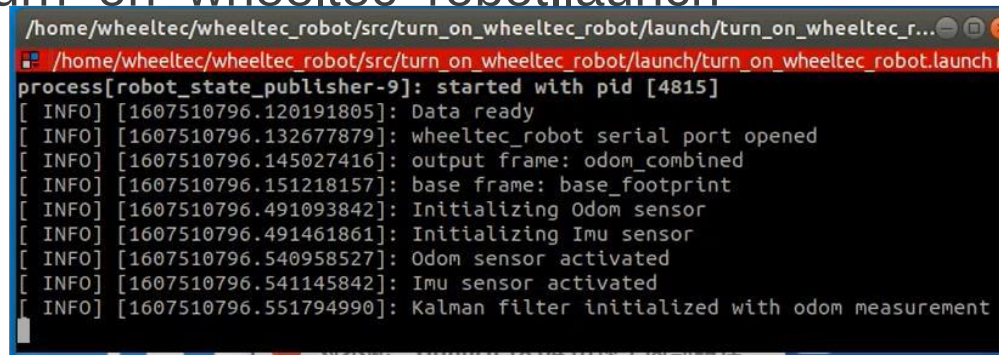
4. In Ubuntu VM, open a terminal (ctrl+alt+t) and type the following

$ sudo apt-get install openssh-server

$ ssh –Y wheeltec@192.168.0.100

$ roslaunch turn_on_wheeltec_robot turn_on_wheeltec_robot.launch



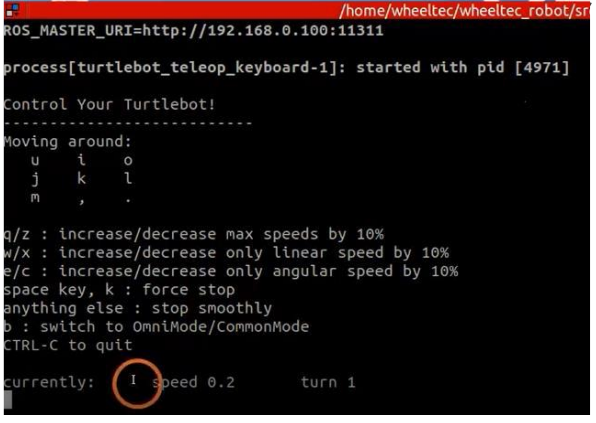


Steps

5. Open another terminal (ctrl+alt+t) and type the following – DO NOT CLOSE THE PREVIOUS TERMINAL

$ ssh –Y wheeltec@192.168.0.100

$ roslaunch wheeltec_robot_rc keyboard_teleop.launch



Changing the car parameters - Demonstration

PID controller

STM32

source code

Kp

Ki

system.c* robot_select_init.c

```
16  Update£°2021-01-29
17
18  All rights reserved
19  ************************************************/
20
21  #include "system.h"
22
23  //Robot software fails to flag bits
24  //»úÆ÷ÈËÊi¼þÊ§ÄÜ±êÖ¾Î»
25  u8 Flag_Stop=1;
26
27  //The ADC value is variable in segments, depending on the number of car models. Currently there are 6 car models
28  //ADCÖµ·Ö¶Î±äÁ¿£¬È¡¾öÓÚÐ¡µÐÍ°ÅÊýÁ¿£¬Ä¿Ç°ÓÐ6ÖÖÐ¡µÐÍ°Å
29  int Divisor_Mode;
30
31  // Robot type variable
32  //»úÆ÷ÈËÐÍ°Å±äÁ¿
33  //0=Mec_Car£¬1=Omni_Car£¬2=Akm_Car£¬3=Diff_Car£¬4=FourWheel_Car£¬5=Tank_Car
34  u8 Car_Mode=0;
35
36  //Servo control PWM value, Ackerman car special
37  //¶æ»ú¿ØÖÆPWMÖµ£¬°¢¿ÆÂüÐ¡³µ×¨ÓÃ
38  int Servo;
39
40  //Default speed of remote control car, unit: mm/s
41  //Ò£¿ØÐ¡³µµÄÄ¬ÈÏËÙ¶È£¬µ¥Î»£°mm/s
42  float RC_Velocity=1000;
43
44  //Vehicle three-axis target moving speed, unit: m/s
45  //Ð¡³µÈýÖáÄ¿±êÔË¶¯ËÙ¶È£¬µ¥Î»£°m/s
46  float Move_X, Move_Y, Move_Z;
47
48  //PID parameters of Speed control
49  //ËÙ¶È¿ØÖÆPID ²ÎÊý
50  float Velocity_KP=300,Velocity_KI=300;
51
52  //Smooth control of intermediate variables, dedicated to omni-directional moving cars
53  //Æ½»¬¿ØÖÆÖÐ¼ä±äÁ¿£¬È«ÏòÔÆ¶¯Ð¡³µ×¨ÓÃ
54  Smooth_Control smooth_control;
55
56  //The parameter structure of the motor
57  //µç»úµÄ²ÎÊý½á¹¹Ìå
58  Motor_parameter MOTOR_A,MOTOR_B,MOTOR_C,MOTOR_D;
59
60  /************ Ð¡³µPID²ÎÊý³õÊ¼»¯ **************************/
```