3. Install the Rosmaster driver library

3.1. Declaration before installing the driver library

The latest driver library has been installed in the factory mirror system of the car, so there is no need to install it repeatedly. If you are not using the factory image, or if the driver library has updated content, you only need to install the driver library.

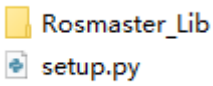
The driver library storage path that comes with the factory system: ~/Software/py_install

For the method of installing the driver library, please refer to the following steps. Here, the installation of version V1.5.8 is used as an example.

3.2. Download the Python driver library file

The latest version of the Rosmaster Python driver library is available in this course material, named py_install.zip.

The compressed package contains the following files:

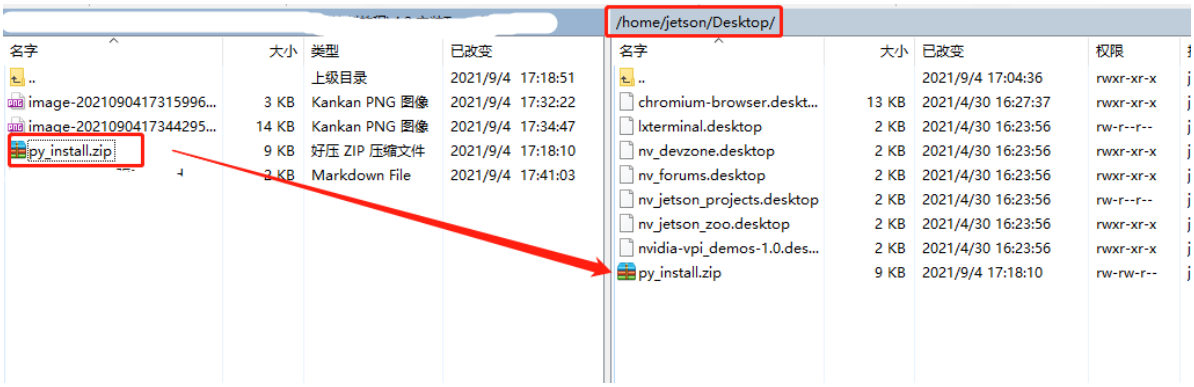


3.3. Transfer files to Jetson Nano

If you use the browser that comes with Jetson Nano to download the file, please download the file to a user-operable path, such as the desktop.

If you use the driver library zip file in the data, or download the driver library file with a computer browser, you can drag the driver library zip file into the Jetson Nano desktop through WinSCP software.

After the installation is successful, the driver library file can be deleted.



3.4. start the installation

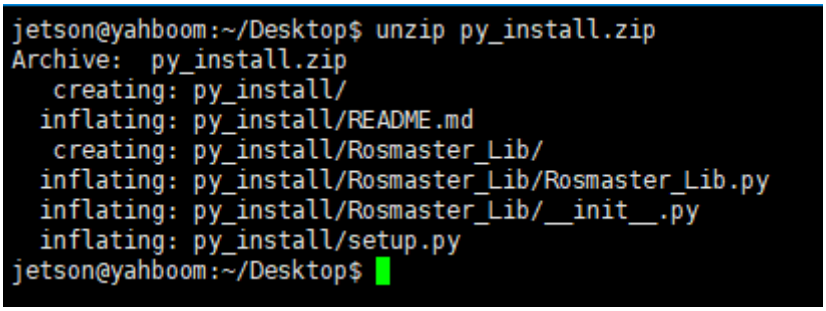
Open the terminal of Jetson Nano and enter the following command to unzip it.

Go to the desktop and check if the file exists, the red box is the target file

```
cd  ~/Desktop &&  ls
```

unzip files

```
unzip py_install.zip
```



Note: The entire documentation routine takes the py_install.zip compressed package placed on the desktop of the Jetson Nano system as an example. If you store the compressed package in a different path, please enter the corresponding directory according to the actual path to operate.

Go to the driver library folder

```
cd  py_install
```

Run the installation command and see the installation version number indicated at the end, indicating that the installation is successful. This command will overwrite the Rosmaster_Lib driver library that has been installed before.

```
sudo  python3 setup.py install
```

```
jetson@yahboom:~/Desktop/py_install$ sudo python3 setup.py install
running install
/home/jetson/.local/lib/python3.6/site-packages/setuptools/command/install.py:37: SetuptoolsDeprecationWarn
ing: setup.py install is deprecated. Use build and pip and other standards-based tools.
  setuptools.SetuptoolsDeprecationWarning,
/home/jetson/.local/lib/python3.6/site-packages/setuptools/command/easy_install.py:159: EasyInstallDeprecat
ionWarning: easy_install command is deprecated. Use build and pip and other standards-based tools.
  EasyInstallDeprecationWarning,
/home/jetson/.local/lib/python3.6/site-packages/pkg_resources/__init__.py:119: PkgResourcesDeprecationWarni
ng: 0.18ubuntu0.18.04.1 is an invalid version and will not be supported in a future release
  PkgResourcesDeprecationWarning,
running bdist_egg
running egg_info
```

3.5. import library file

The name of the Rosmaster driver library is Rosmaster_Lib, and Rosmaster_Lib is used in the program to import the library.

```
from  Rosmaster_Lib  import  Rosmaster
```

3.6. Basic usage of driver library

Check out the video tutorial for this lesson for details.

Source code path: Rosmaster/Samples/3.test_rosmaster.ipynb

3.7. Introduction to API

The following is an introduction to the API of the driver library, and the function usage and parameter content will be introduced in the control course later.

```
|  __del __ (self)
|
 |  __init__(self, car_type=1, com='/dev/myserial', delay=0.002, debug=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
|
 |  clear_auto_report_data(self)
| # Clear the cached data automatically sent by the microcontroller
 |      # Clear the cache data automatically sent by the MCU
|
 |  create_receive_threading(self)
| # Start a thread for receiving and processing data
 |      # Start the thread that receives and processes data
|
 |  get_accelerometer_data(self)
| # Get the three-axis data of the accelerometer, return a_x, a_y, a_z
 |      # Get accelerometer triaxial data, return a_x, a_y, a_z
|
 |  get_battery_voltage(self)
| # Get the battery voltage value
 |      # Get the battery voltage
|
 |  get_gyroscope_data(self)
```

```
| # Get the three-axis data of the gyroscope, return g_x, g_y, g_z
 |      # Get the gyro triaxial data, return g_x, g_y, g_z
|
 |  get_magnetometer_data(self)
| # Get the magnetometer three-axis data, return m_x, m_y, m_z
|
 |  get_motion_data(self)
| # Get the speed of the car, val_vx, val_vy, val_vz
 |      # Get the car speed, val_vx, val_vy, val_vz
|
 |  get_motion_pid(self)
| # Get the motion PID parameters of the car, return [kp, ki, kd]
 |      # Get the motion PID parameters of the dolly and return [kp, ki, kd]
|
 |  get_motor_encoder(self)
| # Get four-way motor encoder data
 |      # Obtain data of four-channel motor encoder
|
 |  get_uart_servo_angle(self, s_id)
| # Read the angle of the bus servo, s_id indicates the ID number of the servo to
be read, s_id=[1-6]
 |      # Read the Angle of the bus steering gear, s_id indicates the ID number
of the steering gear to be
read, s_id=[1-6]
|
 |  get_uart_servo_angle_array(self)
| # Read the angles of six servos at one time [xx, xx, xx, xx, xx, xx], if a
servo is wrong, the bit is -1
 |      # Read the angles of three steering gear [xx, xx, xx, xx, xx, xx] at one
time. If one steering gear
 is wrong, that one is -1
|
 |  get_uart_servo_value(self, servo_id)
| # Read the bus servo position parameter, servo_id=[1-250], return: read ID,
current position parameter
 |      # Read bus servo position parameters, servo_id=[1-250], return: read ID,
current position parameter
s
|
 |  get_version(self)
| # Get the version number of the underlying MCU, such as 1.1
 |      # Get the underlying microcontroller version number, such as 1.1
|
 |  reset_flash_value(self)
| # Reset the data saved in the car's flash and restore the factory default
values.
 |      # Reset the car flash saved data, restore the factory default value
|
 |  set_auto_report_state(self, enable, forever=False)
| # The MCU automatically returns the data status bit, the default is ON, if it
is set to OFF, it will affect part of the data read function.
| # enable=True, the underlying expansion board will send a packet of data every
10 milliseconds, a total of four packets of different data, so each packet of
data is refreshed every 40 milliseconds.  e
nable=False, do not send.
```

```
| # forever=True for permanent storage, =False for temporary use.
 |      # The MCU automatically returns the data status bit, which is enabled by
default. If the switch is
closed, the data reading function will be affected.
 |      # enable=True, The underlying expansion board sends four different
packets of data every 10 millise
conds, so each packet is refreshed every 40 milliseconds.
 |      # If enable=False, the report is not sent.
 |      # forever=True for permanent, =False for temporary
|
 |  set_beep(self, on_time)
| # Buzzer switch, on_time=0: off, on_time=1: keep ringing,
| # on_time>=10: Automatically turn off after xx milliseconds (on_time is a
multiple of 10).
 |      # Buzzer switch. On_time =0: the buzzer is off. On_time =1: the buzzer
keeps ringing
 |      # On_time >=10: automatically closes after xx milliseconds (on_time is a
multiple of 10)
|
 |  set_car_motion(self, v_x, v_y, v_z)
| # Car motion control, v_x=[-1.0, 1.0], v_y=[-1.0, 1.0], v_z=[-5.0, 5.0]
 |      # Car movement control, v_x = [-1.0, 1.0], v_y = [-1.0, 1.0], v_z =
[-5.0, 5.0]
|
 |  set_car_run(self, state, speed, adjust=False)
| # Control the car to move forward, backward, left, right, etc.
| # state=[0~6],=0 stop,=1 forward,=2 backward,=3 left,=4 right,=5 left,=6 right

| # speed=[-100, 100],=0 to stop.
| # adjust=True to enable the gyroscope to assist the movement direction.  =False
to disable.
 |      # Control the car forward, backward, left, right and other movements.
 |      # State =[0~6],=0 stop,=1 forward,=2 backward,=3 left,=4 right,=5 spin
left,=6 spin right
 |      # Speed =[-100, 100], =0 Stop.
 |      # Adjust =True Activate the gyroscope auxiliary motion direction.  If
=False, the function is disab
led.
|
 |  set_car_type(self, car_type)
| # Set the car type
 |      # Set car Type
|
 |  set_colorful_effect(self, effect, speed=255, parm=255)
| # RGB programmable light strip special effects display.
| # effect=[0, 6], 0: stop light effect, 1: running water light, 2: marquee
light, 3: breathing light, 4: gradient light, 5: starlight, 6: battery display
| # speed=[1, 10], the smaller the value, the faster the speed changes.
| # parm, optional, as an additional parameter. Usage 1: Enter [0, 6] for the
breathing light effect to modify the color of the breathing light.
 |      # RGB programmable light band special effects display.
 |      # Effect =[0, 6], 0: stop light effect, 1: running light, 2: running
horse light, 3: breathing ligh
t, 4: gradient light, 5: starlight, 6: power display
 |      # Speed =[1, 10], the smaller the value, the faster the speed changes
```

```
 |      # Parm, left blank, as an additional argument.  Usage 1: The color of
breathing lamp can be modifie
d by the effect of breathing lamp [0, 6]
|
 |  set_colorful_lamps(self, led_id, red, green, blue)
| # RGB programmable light strip control, which can be controlled individually or
as a whole. Before the control, you need to stop the RGB light effects.
| # led_id=[0, 13], control the corresponding number of RGB lights; led_id=0xFF,
control all lights.
| # red,green,blue=[0, 255], representing the color RGB value.
 |      # RGB programmable light belt control, can be controlled individually or
collectively, before contr
ol need to stop THE RGB light effect.
 |      # Led_id =[0, 13], control the CORRESPONDING numbered RGB lights;  Led_id
=0xFF, controls all light
s.
 |      # Red,green,blue=[0, 255], indicating the RGB value of the color.
|
 |  set_motor(self, speed_1, speed_2, speed_3, speed_4)
| # Controls the motor PWM pulses, thus controlling the speed (not using the
encoder to measure the speed).  speed_X=[-100, 100]
 |      # Control PWM pulse of motor to control speed (speed measurement without
encoder). speed_X=[-100, 1
00]
|
 |  set_pid_param(self, kp, ki, kd, forever=False)
| # PID parameter control will affect the change of the motion speed of the car
controlled by the set_car_motion function. By default it can be left unadjusted.

| # kp ki kd = [0, 10.00], you can enter decimals.
| # forever=True for permanent storage, =False for temporary use.
| # Since the permanent storage needs to be written into the chip flash, the
operation time is long, so add the delay time to avoid the problem of packet loss
caused by the microcontroller.
| # Temporary action has a quick response and is valid for one time, and the data
is no longer maintained after restarting the single chip.
 |      # PID parameter control will affect the set_CAR_motion function to
control the speed change of the
car.  This parameter is optional by default.
,  # KP ki kd = [0, 10.00]
 |      # forever=True for permanent, =False for temporary.
 |      # Since permanent storage needs to be written into the chip flash, which
takes a long time to opera
te, delay is added to avoid packet loss caused by MCU.
 |      # Temporary effect fast response, single effective, data will not be
maintained after restarting th
e single chip
|
 |  set_pwm_servo(self, servo_id, angle)
| # Servo control, servo_id: corresponding to the ID number, angle: corresponding
to the angle value of the servo
 |      # servo_id=[1, 4], angle=[0, 180]
 |      # Servo control, servo_id: corresponding, Angle: corresponding servo
Angle value
|
```

```
 |  set_pwm_servo_all(self, angle_s1, angle_s2, angle_s3, angle_s4)
| # Control the angle of four PWM channels at the same time, angle_sX=[0, 180]
 |      # At the same time control four PWM Angle, angle_sX=[0, 180]
|
 |  set_uart_servo(self, servo_id, pulse_value, run_time=500)
| # Control the bus servo.  servo_id:[1-255], indicates the ID number of the
servo to be controlled, when id=254, it controls all connected servos.
| # pulse_value=[96,4000] indicates the position to which the servo should run.
| # run_time indicates the running time (ms), the shorter the time, the faster
the servo rotates. Minimum is 0, maximum is 2000
 |      # Control bus steering gear.  Servo_id :[1-255], indicating the ID of the
steering gear to be contr
olled. If ID =254, control all connected steering gear.
 |      # pulse_value=[96,4000] indicates the position to which the steering gear
will run.
 |      # run_time indicates the running time (ms). The shorter the time, the
faster the steering gear rota
tes.  The minimum value is 0 and the maximum value is 2000
|
 |  set_uart_servo_angle(self, s_id, s_angle, run_time=500)
| # Set the bus servo angle interface: id:7-9, angle: 7:[0, 225], 8:[30, 270], 9:
[30, 180], set the angle to which the servo should move
.
| # Set the vertical and upward clamping state, the three servos are all 180
degrees, the 7/8th turn clockwise (down) to decrease, counterclockwise (up) to
increase, and the clamp is released
To reduce, clamp to increase.
| # run_time indicates the running time (ms), the shorter the time, the faster
the servo rotates. Minimum is 0, maximum is 2000
 |      # Set bus steering gear Angle interface: ID :7-9, Angle :7 :[0, 225], 8:
[30, 270], 9:[30, 180], set
 steering gear to move to the Angle.
 |      # Set up the vertical clamping state, the three steering gear are 180
degrees, 7/8 clockwise (down)
 to decrease, counterclockwise (up) to increase, clip release to decrease,
clamping to increase.
 |      # run_time indicates the running time (ms). The shorter the time, the
faster the steering gear rota
tes.  The minimum value is 0 and the maximum value is 2000
|
 |  set_uart_servo_angle_array(self, angle_s=[90, 90, 90, 90, 90, 180],
run_time=500)
| # Control the angles of all servos of the robotic arm at the same time.
 |      # Meanwhile, the Angle of all steering gear of the manipulator is
controlled
|
 |  set_uart_servo_id(self, servo_id)
| # Set the ID number of the bus servo, servo_id=[1-250].
| # Please make sure to connect only one bus servo before running this function,
otherwise all connected bus servos will be set to the same ID, resulting in
control confusion
.
 |      # Set the bus servo ID, servo_id=[1-250].
 |      # Before running this function, please confirm that only one bus actuator
is connected. Otherwise,
```

```
all connected bus actuators will be set to the same ID, resulting in confusion of
control
|
 |  set_uart_servo_offset(self, servo_id)
| # Set the median deviation of the robotic arm, servo_id=0~6, =0 all restore the
factory default value
 |      # Run the following command to set the mid-bit deviation of the
manipulator: servo_id=0 to 6, =0 Re
store the factory default values
|
 |  set_uart_servo_torque(self, enable)
| # Turn off/on bus servo torque, enable=[0, 1].
| # enable=0: Turn off the steering gear torque, you can turn the steering gear
by hand, but the command cannot control the rotation;
| # enable=1: Turn on the torque force, the command can control the rotation, and
the servo cannot be turned by hand.
 |      # Turn off/on the bus steering gear torque force, enable=[0, 1].
 |      # enable=0: Turn off the torque force of the steering gear, the steering
gear can be turned by hand
, but the command cannot control the rotation;
 |      # enable=1: Turn on torque force, command can control rotation, can not
turn steering gear by hand
```