6.1 Basic movement of Jetbot Mini

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP that starts automatically after booting, execute the following command:

```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP function, execute the following command:

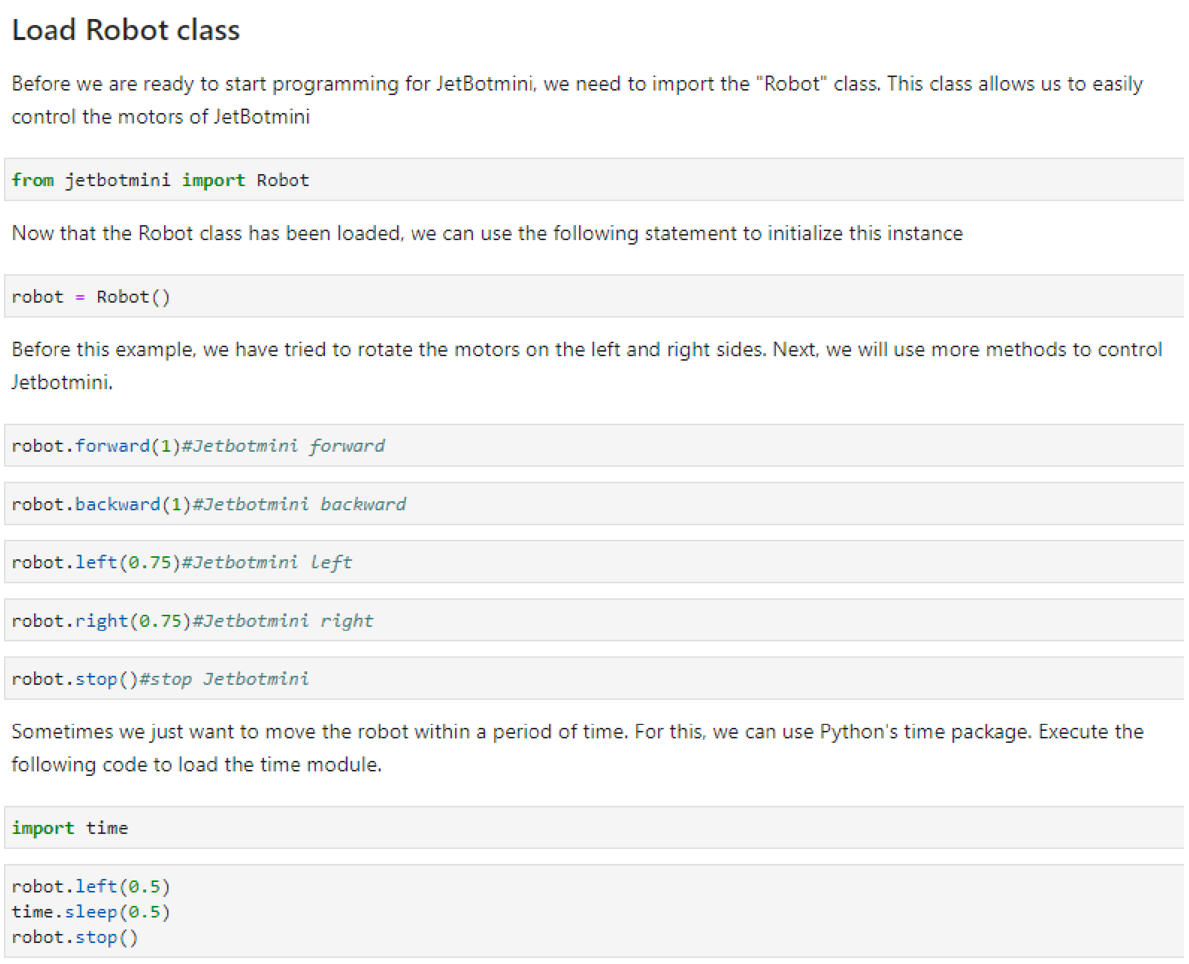
```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

6.1.1 Create a robot instance to start your Jetbotmini

In the previous course, we have initially tried to make some classified applications through AI, and used all the driver packages of Jetbotmini to control the corresponding hardware on it. From this section on, we will officially start the AI integrated application to integrate our Jetbotmini gradually becomes a smarter robot step by step, so let's do it now!

In the previous course, we created Jetbotmini objects and used some of the methods to control some of Jetbotmini's hardware, such as individually controlling the rotation of the left and right motors. In fact, Jetbotmini also packs many methods to directly control the movement of Jetbotmini. Let’s take a look. Review some of the methods used in previous courses and discover more new ways to control the movement of Jetbotmini:

It is still the old rule before starting to use it, first import the Jetbotmini package and then create a robot instance that controls the movement of Jetbotmini, we can easily control Jetbotmini.



Load Robot class

Before we are ready to start programming for JetBotmini, we need to import the "Robot" class. This class allows us to easily control the motors of JetBotmini

```python
from jetbotmini import Robot
```

Now that the Robot class has been loaded, we can use the following statement to initialize this instance

```python
robot = Robot()
```

Before this example, we have tried to rotate the motors on the left and right sides. Next, we will use more methods to control Jetbotmini.

```python
robot.forward(1)#Jetbotmini forward
```

```python
robot.backward(1)#Jetbotmini backward
```

```python
robot.left(0.75)#Jetbotmini left
```

```python
robot.right(0.75)#Jetbotmini right
```

```python
robot.stop()#stop Jetbotmini
```

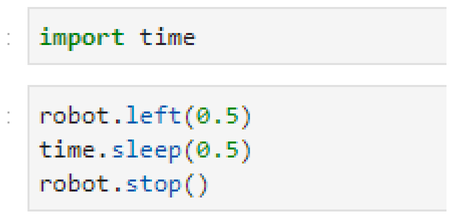
Sometimes we just want to move the robot within a period of time. For this, we can use Python's time package. Execute the following code to load the time module.

```python
import time
```

```python
robot.left(0.5)
time.sleep(0.5)
robot.stop()
```

After running the above code, we should be able to see Jetbotmini move forward and backward at a maximum speed of 1, turn left and right at a speed of 0.75, and stop Jetbotmini.

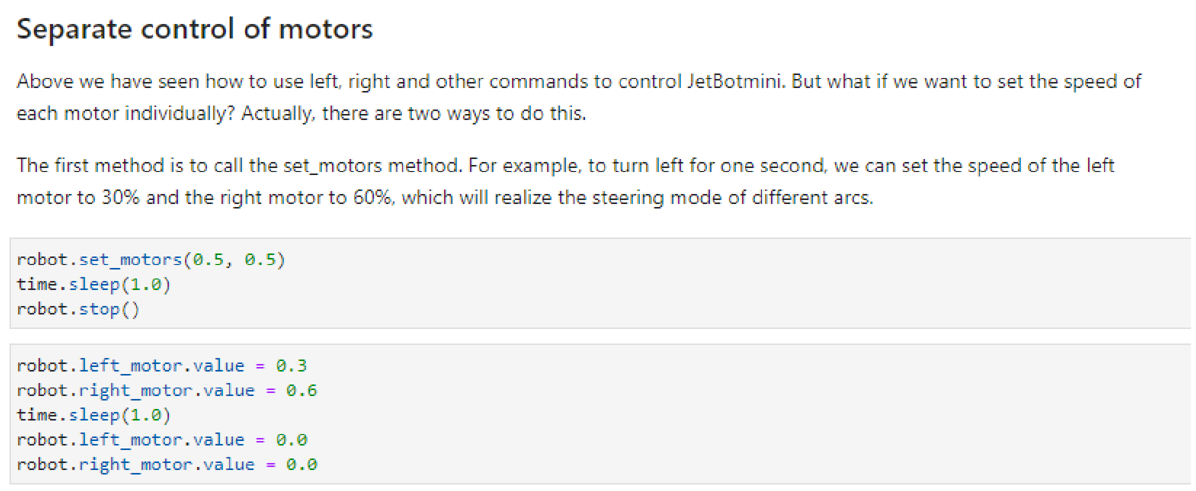
Sometimes we only want to move the robot within a period of time. For this, we can use Python's time package. The time module is often used to save system resources when creating a new thread to handle some small tasks. We will execute the following cell code Seeing Jetbotmini turn left, it stopped.



```python
import time
```

```python
robot.left(0.5)
time.sleep(0.5)
robot.stop()
```

In addition to setting the value of `robot.left_motor.value` used in the previous courses to control the motor individually, there is also `robot.set_motors(Left_Motor_Value, Right_Motor_Value)` method. They achieve the same effect, but `robot.set_motors (Left_Motor_Value, Right_Motor_Value)` method is used a bit more, because you can set the speed of the two motors at one time, which is more convenient:



Separate control of motors

Above we have seen how to use left, right and other commands to control JetBotmini. But what if we want to set the speed of each motor individually? Actually, there are two ways to do this.

The first method is to call the set_motors method. For example, to turn left for one second, we can set the speed of the left motor to 30% and the right motor to 60%, which will realize the steering mode of different arcs.

```
robot.set_motors(0.5, 0.5)
time.sleep(1.0)
robot.stop()
```

```
robot.left_motor.value = 0.3
robot.right_motor.value = 0.6
time.sleep(1.0)
robot.left_motor.value = 0.0
robot.right_motor.value = 0.0
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/8.Control_jetbotmini_motion/Control_jetbotmini_motion.ipynb

6.1.2 Comprehensive interactive use of ipywidgets controls and traitlets

Before using ipywidgets and traitlets, let’s get to know them briefly.

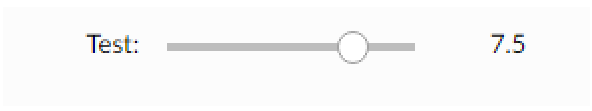
ipywidgets:

Those who have written APP will definitely be familiar with widgets. As the name suggests, it means widgets/widgets. In the interactive interface between APP and us, most of us use it to complete the interaction, and what we want to use now Ipywidgets, if you have ever created a graphical user interface (GUI), then you already know what a widget is. But let's quickly define:

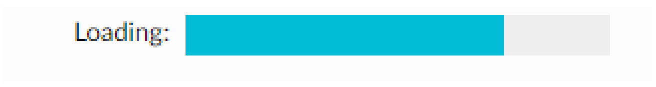
ipywidgets widgets are GUI elements, such as buttons, drop-down menus or text boxes, which reside in the browser, allowing us to control code and data by responding to events and calling designated handlers. Each widget has its own specificity. The attributes to deal with some data information.

for example:

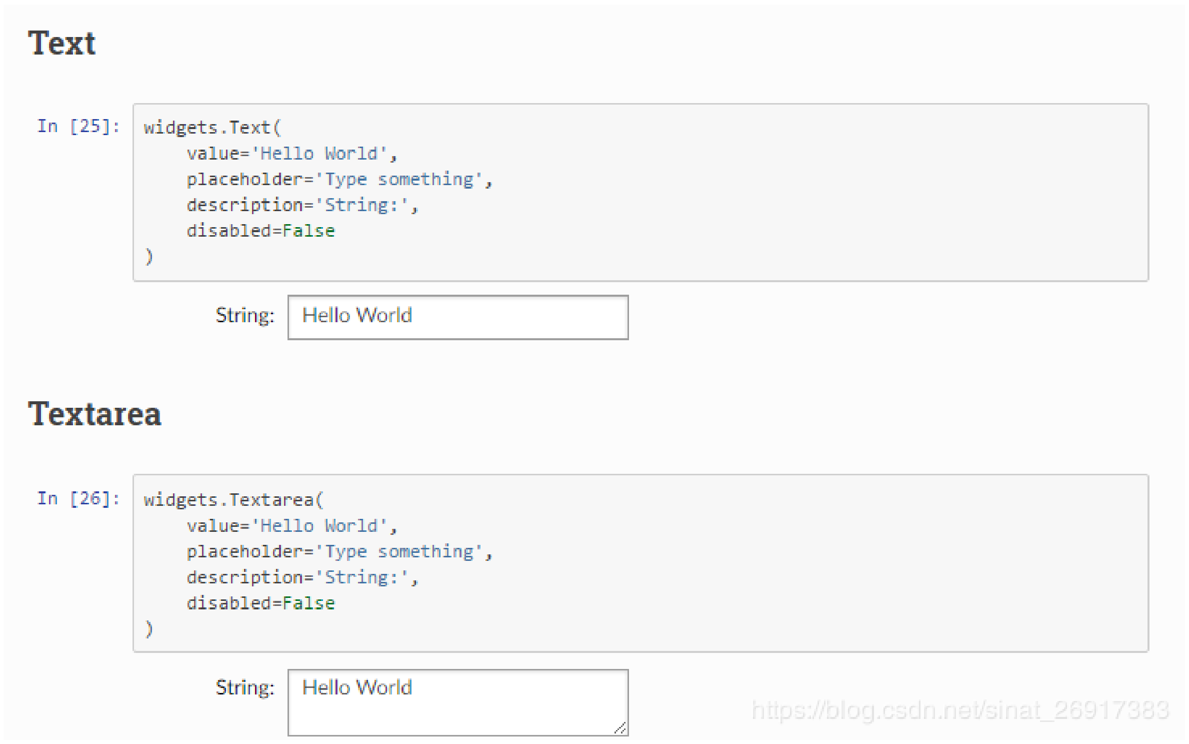
Slider



Progress bar



Text display component



Text

```
In [25]: widgets.Text(
             value='Hello World',
             placeholder='Type something',
             description='String:',
             disabled=False
         )
```

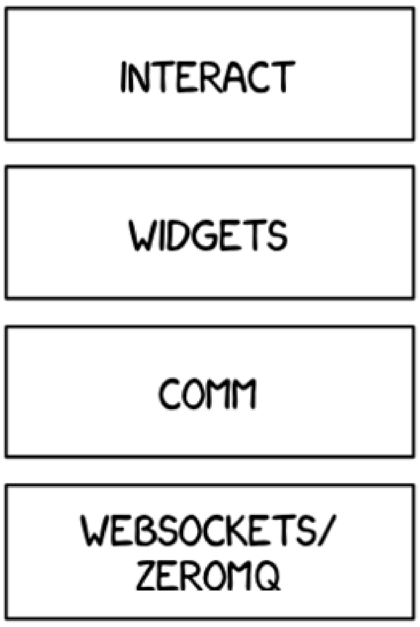
String: Hello World

Textarea

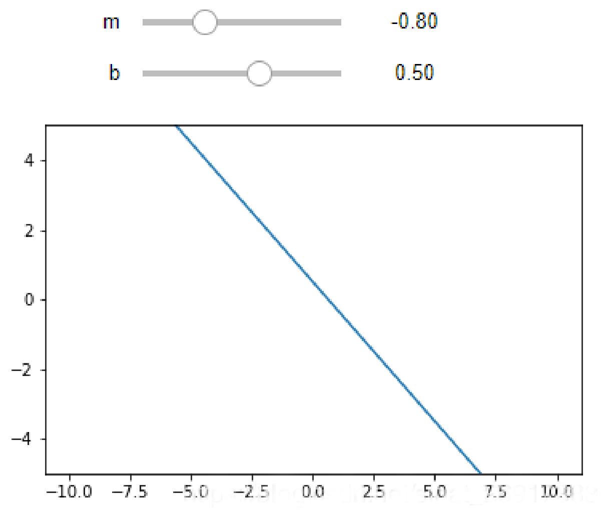
```
In [26]: widgets.Textarea(
             value='Hello World',
             placeholder='Type something',
             description='String:',
             disabled=False
         )
```

String: Hello World

Picture display component



Data visualization with graphics and sliders



In addition to some of the components listed above, ipywidgets also has a lot of component tools that are super convenient to use because the length is not listed one by one, waiting for us to slowly explore and discover in future use!

traitlets:

Traitlets is a framework that allows Python classes to have type checking, dynamically calculated default values, and "on change" callback properties.

The package also includes a mechanism for configuration using traitlets, loading values from files or command line parameters. This is the unique layer on top of the traitlet, so you can use the traitlet in your code without using the configuration mechanism.

When we debug Jetbotmini robots, we bind components and data together through ipywidgets and traitlets, which can be called back and refreshed in real time in the background, which can greatly improve our debugging efficiency. I think this is like an artifact!

Let's make preliminary use of them step by step:

First import the relevant package, create two sliders, define its properties when creating the sliders.

Use traitlets library connected to control operation of the motor Jupyter lab HTML

Next, I will introduce a very cool feature, that is, in Jupyter Notbooks, we can make some small graphical buttons (controls) on this page, and use traitlets to connect these widgets for control operations. In this way, we can control our car through the buttons on the web page, which will become very convenient and fun.

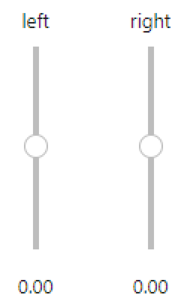
To illustrate how to write a program, we first create and display two sliders for controlling the motor.

```
import ipywidgets.widgets as widgets
from IPython.display import display

# create two sliders with range [-1.0, 1.0]
left_slider = widgets.FloatSlider(description='left', min=-1.0, max=1.0, step=0.01, orientation='vertical')
right_slider = widgets.FloatSlider(description='right', min=-1.0, max=1.0, step=0.01, orientation='vertical')

# create a horizontal box container to place the sliders next to eachother
slider_container = widgets.HBox([left_slider, right_slider])

# display the container in this cell's output
display(slider_container)
```



After running the above code, you should see two vertical sliders displayed on it.

Tips: In Jupyter Lab, you can actually pop up cells to other windows, such as these two sliders. Although it is not in the same window, it is still connected to this notebook. The specific operation is to move the mouse to the cell and right click (for example: slider), select "Create new view for output" (create a new window for output), and then drag the window to the place you are satisfied with.

Try to click and drag the slider up and down, you will see the value change. Please note that currently, Jetbotmini's motors do not respond when we move the slider, that's because we haven't connected them to the motors yet! Below we will achieve this by using the link function in the traitlets package.

```
import traitlets
left_link = traitlets.link((left_slider, 'value'), (robot.left_motor, 'value'))
right_link = traitlets.link((right_slider, 'value'), (robot.right_motor, 'value'))
```

Now try to drag the slider (slowly drag it first, so as not to cause damage when your Jetbotmini suddenly rushes out of the border), you should see the corresponding motor rotating!

The link function we created above actually creates a two-way link! That means, if we set the motor value elsewhere, the slider will update accordingly! Try to execute the following code block:

```
robot.forward(0.5)
time.sleep(1.0)
robot.stop()
```

After execution, you should see that the slider has also changed, responding to the speed value of the motor. If we want to disconnect this connection, we can call the unlink method to disconnect one by one.

```
left_link.unlink()
right_link.unlink()
```

After disconnecting, they return to the state where there was no connection at the beginning, but if we don't want a two-way connection, for example, we only want to use a slider to display the speed value of the motor, but not for control, then To achieve this function, we can use the dlink function, with the source on the left and the target on the right (the data comes from the motor, and then it must be displayed on the target).

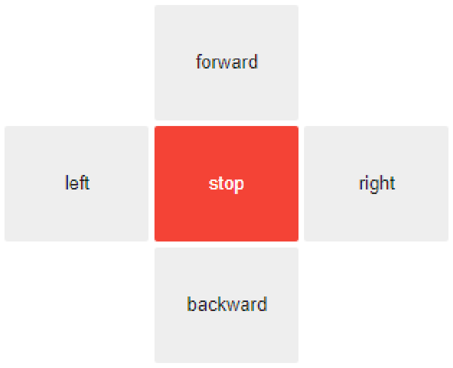
```
left_link = traitlets.dlink((robot.left_motor, 'value'), (left_slider, 'value'))
right_link = traitlets.dlink((robot.right_motor, 'value'), (right_slider, 'value'))
```

Another way to use traitlets that I want to introduce is to attach functions to events (for example, forward). As long as there is a change to the object, the function will be called, and some information about the change will be passed, such as the old value and the new value.

Let us first create some buttons to control the robot to display on the notebook:

```
# Create button
button_layout = widgets.Layout(width='100px', height='80px', align_self='center')
stop_button = widgets.Button(description='stop', button_style='danger', layout=button_layout)
forward_button = widgets.Button(description='forward', layout=button_layout)
backward_button = widgets.Button(description='backward', layout=button_layout)
left_button = widgets.Button(description='left', layout=button_layout)
right_button = widgets.Button(description='right', layout=button_layout)

# Display button
middle_box = widgets.HBox([left_button, stop_button, right_button], layout=widgets.Layout(align_self='center'))
controls_box = widgets.VBox([forward_button, middle_box, backward_button])
display(controls_box)
```



After executing the above cell code block, you should see a set of robot control buttons shown above, but now you click the button and nothing will happen. To achieve control, we need to create some functions attached to the button on_click event

```python
def stop(change):
    robot.stop()

def step_forward(change):
    robot.forward(0.8)
    time.sleep(0.5)
    robot.stop()

def step_backward(change):
    robot.backward(0.8)
    time.sleep(0.5)
    robot.stop()

def step_left(change):
    robot.left(0.6)
    time.sleep(0.5)
    robot.stop()

def step_right(change):
    robot.right(0.6)
    time.sleep(0.5)
    robot.stop()
```

Now that we have defined those functions, let's attach these functions to the on_click event of each button

```python
# link buttons to actions
stop_button.on_click(stop)
forward_button.on_click(step_forward)
backward_button.on_click(step_backward)
left_button.on_click(step_left)
right_button.on_click(step_right)
```

After executing the above cell code, we can now control Jetbotmini to move forward, backward, turn left, turn right and stop by clicking the button component we created above.

Then, in order to prevent Jetbotmini from being out of control and causing damage when the signal of the connected Jetbotmini is disconnected, Jetbotmini also provides a method to solve this problem, that is, the heartbeat switch function:

The following cell code block will introduce how to use the 'heartbeat' package to stop Jetbotmini's movement. This is an easy way to check if the connection between Jetbotmini and the browser still exists. The heartbeat cycle (in seconds) can be adjusted by the slider created and displayed after executing the cell code. If the browser cannot communicate back and forth within two heartbeats, then the'status' attribute of the heartbeat The value will be set to dead, and once the connection is restored, the status attribute will be set to alive.

```python
from jetbotmini import Heartbeat

heartbeat = Heartbeat()

# this function will be called when heartbeat 'alive' status changes
def handle_heartbeat_status(change):
    if change['new'] == Heartbeat.Status.dead:
        robot.stop()

heartbeat.observe(handle_heartbeat_status, names='status')

period_slider = widgets.FloatSlider(description='period', min=0.001, max=0.5, step=0.01, value=0.5)
traitlets.dlink((period_slider, 'value'), (heartbeat, 'period'))

display(period_slider, heartbeat.pulseout)
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/8.Control_jetbotmini_motion/Control_jetbotmini_motion.ipynb