5.1 Basic movement of Jetbot

1. Create a robot instance

We created the Jetbot object in the previous course. We used some methods to control some of Jetbot's hardware, such as controlling the rotation of the left and right motors separately, controlling the LED light under the LOGO to produce breathing effects, etc. In fact, Jetbot is also packed with many methods of controlling Jetbot robot car directly .

Let's review some of the methods used in the previous lesson and discover more new ways to control Jetbot motion:

We can easily control the Jetbot by importing the Jetbot package and then creating a robot instance that controls the Jetbot robot car movement.

Loading the Robot class

Before start programming for JetBot, we need to import the "Robot" class. This class allows us to easily control the JetBot motor

```
from jetbot import Robot
```

Robot class has been loaded, we can initialize this instance with the following statement.

```
robot = Robot()
```

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

```
robot.down(1)#PTZ rise
```

```
robot.up(1)#PTZ decline
```

```
robot.vertical_motors_stop()#stop PTZ
```

```
robot.forward(1)#Jetbot Robot car advance
```

```
robot.backward(1)#Jetbot Robot car back
```

```
robot.left(0.75)#Jetbot turn left
```

```
robot.right(0.75)#Jetbot turn right
```

Yahboom

```
robot.stop()#stop Jetbot
```

```
robot.set_bln(1)#Set Jetbot's onboard breathing lamp brightness to 1 (value range: 0 - 1.0)
```

After running the above code, we can see the Jetbot robot car pan/tilt rise, the pan/tilt down, the pan/tilt stop, the Jetbot robot car advance and back with a maximum speed 1, turn left and turn right with 0.75, stop. And we can see that brightness the LED light under LOGO of is 1.

If we want to move the robot over a period of time. At this point, we can use Python's time package.

The time module is often used to save system resources when the new thread handles some small tasks.

After we execute the following code, we will see that Jetbot robot car turns left and stop.

Code as shown below:

```
import time

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


About control the motor separately.
In addition to the **robot.left_motor.value** to set the value in the previous lesson, as well as robot.set_motors**(Left_Motor_Value, Right_Motor_Value)**, they achieve the same effect.

However, the robot.set_motors**(Left_Motor_Value, Right_Motor_Value)** method is more convenient because the speed of the two sides of the motor can be set at the same time.
Code as shown below:

Individually controlling the motor

Set the speed of each motor separately: The first method is to call set_motors. For example, turn left for one second. We can set the left motor speed to 30% and the right motor to 60%, which will achieve different arc steering modes.

```
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
```

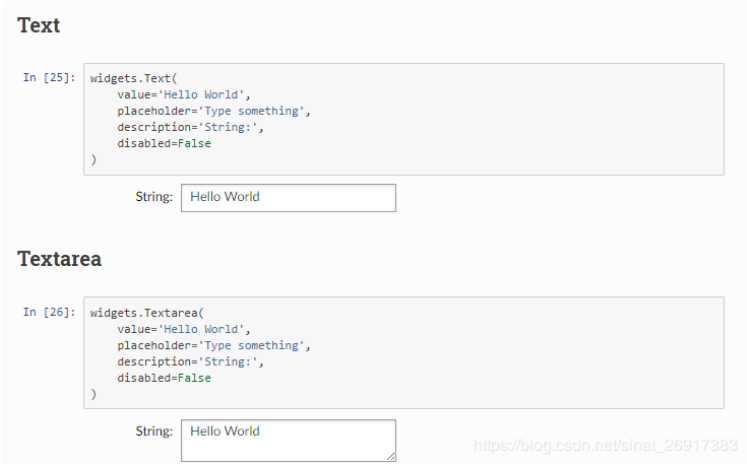
The corresponding complete source code is located at:
/home/jetbot/Notebook/9.Basic movement of Jetbot/Basic movement of Jetbot.ipynb

2. Integrated interaction between ipywidgets control and traitlets
About Ipywidgets:
The ipywidgets widget is a GUI element, such as a button, drop-down menu, or text box, that resides in the browser, allowing us to control code and data by responding to events and calling the specified handler, each widget has its own specific The properties to handle some data information.
such as:

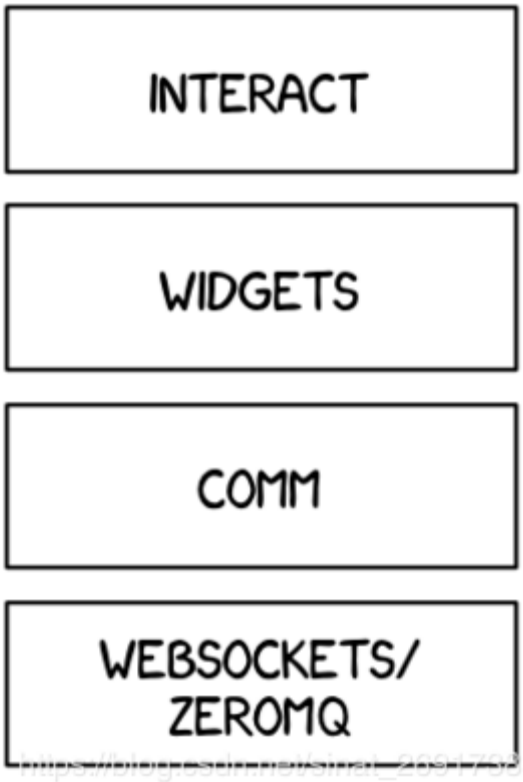
Slider

Progress Bar

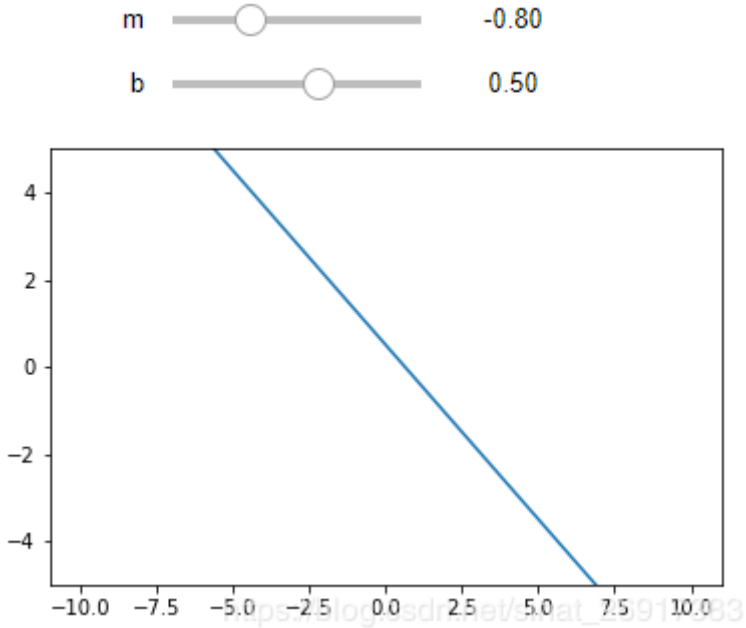
Text display component



Picture display component



Data visualization of graphics plus slider



About traitlets：

Traitlets is a framework that allows Python classes to possess type checking, dynamic calculation defaults, and "on change" callback properties.

The package also includes a mechanism for configuring with traitlets to load values from file or command line arguments.

This is a unique layer at the top of the traitlet, so you can use traitlets in your code without using a configuration mechanism.

When we debug the Jetbot robot, we bind the components and data through ipywidgets and traitlets, and we can call back the refresh in real time in the background, which can greatly improve our debugging efficiency.

How to initial use them:

First import the relevant package, create two sliders, and define its properties when you create the slider.

Code as shown below:

Using the traitlets library to connect to the Jupyter lab HTML control to operate the motor

Next we learn how to make some small graphical buttons (controls) on the Jupyter Notbooks page, then use traitlets to connect these widgets for control. In this way, we can control the car through the buttons on the web page. First, we need to 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 can see two vertical sliders on it.
As shown below:

Tips:
In the Jupyter Lab, you can actually pop the cell into another window, such as the two sliders. Although not in the same window, it is still connected to this notebook. The specific operation is to move the mouse to the right button of the cell (for example: slider), select "Creat new view for output", and then drag the window to the place you are satisfied.

Try clicking and dragging the slider up and down to see the change in value.

!!! Note:
Jetbot's motors are currently not responding when we move the sliders, because we have not connected them to the motor yet!

Below we will use the link function in the traitlets package to achieve. Code as shown below:

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

You can try dragging the slider (slowly drag it first to prevent your Jetbot from suddenly rushing out of the boundary and causing damage), and you can see that the servo attached to the camera is spinning.
The link function we created above actually creates a two-way link!
That means that if we set the motor value elsewhere, the slider will be updated!

For example, the code shown below:

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

After running the above code, you can see that the slider has also changed, and the speed value of the motor will changed. If we want to disconnect this connection, we can call the unlink method to disconnect one by one.
Code shown below:

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

After disconnecting, they return to the state of no contact.

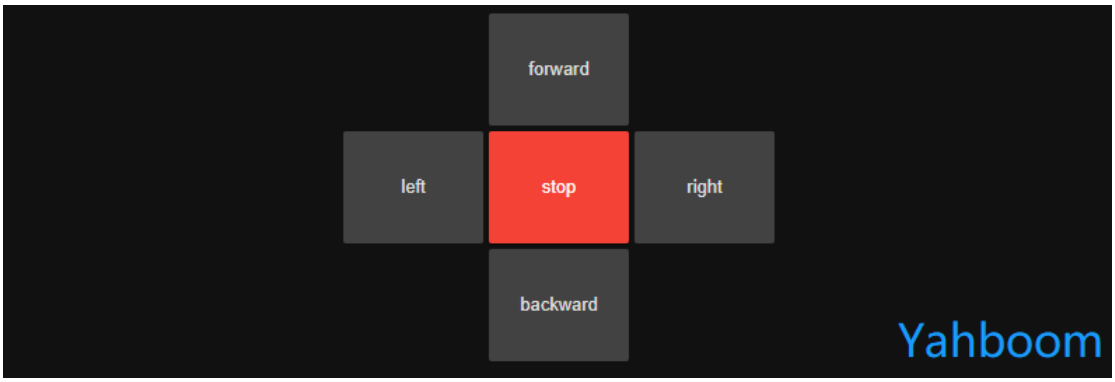
However, if we don't want a two-way connection, for example, we only want to use the slider to display the speed value of the motor, and do not want to control it, then to achieve this function, we can use the **dlink** function, the source on the left, On the right is the target, (the data comes from the motor and then on the target).
The code is shown below:

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

Another method:
Another method to use traitlets is that attach a function to an event (for example, forward). As soon as the object changes, the function is called and some information that has changed is passed, such as the old and new values.
Create some buttons for controlling the robot to display on the notebook:

```
# 创建按钮
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)

# 显示按钮
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 code, you can see a set of robot control buttons shown above, but now you don't have any reaction when you click the button. If we want to start control, we need to create some functions attached to the button on_click event.
The code as shown below:

```
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()
```


We have defined those functions, we will attach these functions to the on_click event of each button.
The code as shown below:

```
# 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 code, you will can control the Jetbot forward, backward, left turn, right turn, and stop by clicking the button component we created above.

In order to prevent the connected Jetbot signal from being disconnected, Jetbot is out of control and the damage is caused. Jetbot also provides a way to solve this problem, which is the heartbeat switch function.

The following code block is about how to use the 'heartbeat' package to stop

Jetbot's movement.

This is a way to check if the connection between Jetbot and the browser still exists. You can adjust the heartbeat period (in seconds) by executing the slider after the code is executed. If the heartbeat cannot communicate back and forth between browsers, the heartbeat 'status' attribute value will be Will be set to dead, once the connection is restored, the status property will be set to alive.

```
from jetbot 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 at: /home/jetbot/Notebook/9.Basic movement of Jetbot/Basic movement of Jetbot.ipynb