

03. Python principles, ROS Publisher, ROS Subscriber

Lecture

Python principles



- Interpreted, high-level programming language
- Name tribute to the comedy group *Monty Python*
- Powerful, still easy to learn, easy to use
- Readability
- **Whitespace indentation**



- Dynamically-typed
- Garbage collector and reference counting
- Object oriented programming
- Used in: AI, web applications, scientific computing, and many other areas

- python3

Python syntax

```
import numpy as np
import math

class A:
    def __init__(self, name):
        self.name = name

    def do_something(self):
        # will do something
        print(self.name + " is doing something.")

    def count_to(self, n):
        # count to n, tell if the number is odd or even
        for i in range(n):
            if i % 2 == 0:
                print(i + ", it's even.")
            else:
                print(i + ", it's odd.")

if __name__ == "__main__":
    a = A("John")
    a.do_something()
    a.count_to(10)
```

Practice

1: Hello, World!

1. Navigate to the `~/catkin_ws/src/ros_course/scripts/` folder and create the file `hello.py` :

```
cd catkin_ws/src/ros_course/scripts
touch hello.py
```

2. Type or copy this line into the file `hello.py` :

```
print("Hello, World!")
```

Tip

In gedit: Fix whitespace handling in gedit: Preferences -> Editor -> Insert spaces instead of tabs.

3. To run the file, `cd` to the `scripts` directory and type:

```
python3 hello.py
```

Tip

In the case of issues with permissions, type the following to grant the file permission to execute: `chmod +x hello.py`

4. Modify the script to replace the word "World" with a command line argument:

```
import sys

msg = sys.argv[1]
print("Hello," , msg, "!")
```

5. Run the file:

```
python3 hello.py John
```

2: Moving the turtle straight

1. Write a ROS node which communicates with the `turtlesim_node` and moves the turtle straight forward until it reaches the given distance. Open a terminal and create the file `turtlesim_controller.py` in the folder `~/catkin_ws/src/ros_course/scripts`:



```
cd catkin_ws/src/ros_course/scripts
touch turtlesim_controller.py
```

2. Add `turtlesim_controller.py` to `CMakeLists.txt` :

```
catkin_install_python(PROGRAMS
  scripts/talker.py
  scripts/listener.py
  scripts/turtlesim_controller.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

3. Copy the skeleton of the program into `turtlesim_controller.py` :

```
import rospy
import math

class TurtlesimController:
    def __init__(self):
        # Call init node only once
        rospy.init_node('turtlesim_controller', anonymous=True)
        # Define publisher here

    def go_straight(self, speed, distance, forward):
        # Implement straght motion here
```

```

if __name__ == '__main__':
    # Init
    tc = TurtlesimController()
    # Send turtle on a straight line
    tc.go_straight(1, 4, True)

```

4. Launch a `turtlesim_node`, then find the topic we can use to control its movement. In three separate terminal windows:

```
roscore
```

```
roslaunch turtlesim turtlesim_node
```

```

rostopic list
rostopic info /turtle1/cmd_vel
rostopic show geometry_msgs/Twist

```

5. Import the message type `geometry_msgs/Twist` and create the publisher handle object for the topic named `turtlesim_controller.py`:

```

from geometry_msgs.msg import Twist

#...

self.twist_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)

```

6. Implement the `go_straight` method. Calculate how much time it takes for the turtle to move to the given distance with the given velocity. Publish and repeat a message to set the velocity, and when the calculated time is up, send another message to set the velocity to 0. A little help on the usage of the API:

```

# Create and publish msg
vel_msg = Twist()
if forward:
    vel_msg.linear.x = speed
else:
    vel_msg.linear.x = -speed
vel_msg.linear.y = 0
vel_msg.linear.z = 0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
vel_msg.angular.z = 0

# Set loop rate
rate = rospy.Rate(100) # Hz

```

```

# Publish first msg and note time
self.twist_pub.publish(vel_msg)
t0 = rospy.Time.now().to_sec()

# Publish msg while the calculated time is up
while (some condition...) and not(rospy.is_shutdown()):
    self.twist_pub.publish(vel_msg)
    # ...and stuff
    rate.sleep() # loop rate

# Set velocity to 0
vel_msg.linear.x = 0
self.twist_pub.publish(vel_msg)

```

7. Launch the node:

```
roslaunch ros_course turtlesim_controller.py
```

3: Drawing polygons



1. Implement a method to turn the turtle with a given angle in `turtlesim_controller.py` in a similar way to the straight movement. `Omega` refers to the angular velocity.

```
def turn(self, omega, angle, forward):  
    # Implement rotation here
```

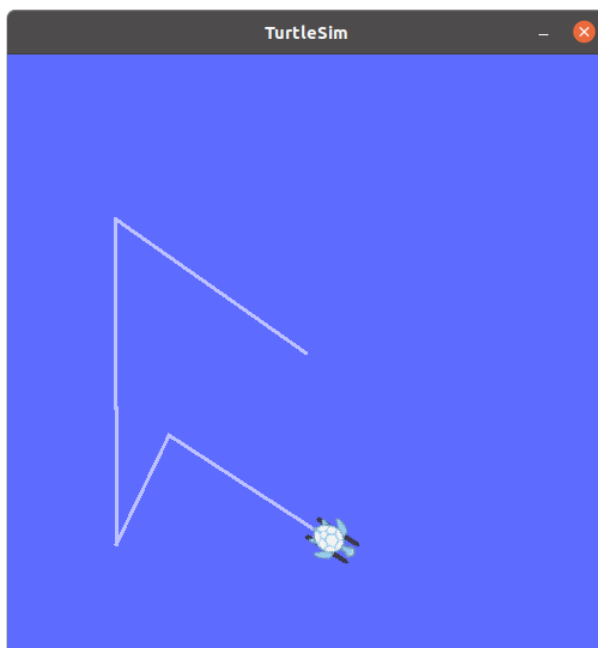
2. Implement a method that draws a square with the turtle. Use the methods `go_straight` and `turn`.

```
def draw_square(self, speed, omega, a):
```

3. Implement a method to draw arbitrary regular polygons.

```
def draw_poly(self, speed, omega, N, a):
```

4: Go to method



1. Search for the topic `turtlesim` which publishes the pose (position and orientation) of the turtle into:

```
rostopic list  
rostopic info /turtle1/pose  
rosmmsg show turtlesim/Pose
```


2. Create a subscriber for the topic and write the callback function:

```
# Imports
from turtlesim.msg import Pose

# Constructor
self.pose_subscriber = rospy.Subscriber('/turtle1/pose', Pose, self.cb_pose)

# New method for TurtlesimController
def cb_pose(self, msg):
    self.pose = msg
```

3. Implement the method `go_to`. Test it by calling from the main.

```
# ...

# Go to method
def go_to(self, speed, omega, x, y):
    # Stuff

# Main
if __name__ == '__main__':
    # Init
    tc = TurtlesimController()
    # 1 sec sleep so subscriber can get msgs
    rospy.sleep(1)
    tc.go_to(1, 2, 2, 8)
    tc.go_to(1, 2, 2, 2)
    tc.go_to(1, 2, 3, 4)
    tc.go_to(1, 2, 6, 2)
```

Bonus exercise: Advanced go to

Write a more accurate go to method using proportional controller.

Useful links

- [For loops in python](#)
- [Some python functions](#)
- [Turtlesim documentation](#)
- [atan2](#)