GoPiGo Python Tutorials

Contents

NOTE: All Python code is compatible with Python 3.5. This is the current version of Python on the GoPiGo.

Go to https://gopigo3.readthedocs.io/en/master/api-basic/easygopigo3.html#easygopigo3 for information on the easygopigo3 library. All example code in this Tutorial is derived from this library.

First Steps

Go to the Code Examples folder in the WNCCNASA GitHub repository. Copy and paste the example code to the GoPiGo to get started and test your GoPiGo.

Easy Movement Tutorial

It is time to do some tutorials to learn the GoPiGo library better.

Learning points

- Functions, Loops, Movement, GoPiGo Blinkers

```python
#!/usr/bin/env python3
"""
    Name: easy_movement.py
    Author: William A Loring
    Created: 09-18-21 Revised:
    Purpose: Demonstrate a sampling of GoPiGo dead reckoning movements
"""
# This uses the EasyGoPiGo3 library.  You can find more information on the li
# here:  https://gopigo3.readthedocs.io/en/master/api-basic/easygopigo3.html#

# Import the time library for the sleep function
import time
# Import GoPiGo3 library
from easygopigo3 import EasyGoPiGo3

# Create an instance of the GoPiGo3 class
# GPG is the GoPiGo3 object used to access methods and properties
gpg = EasyGoPiGo3()


#------------------------- SQUARE RIGHT ------------------------#
def square_right(distance):
    """
        Drive a right square based on the distance argument
    """
    # Loop four times,  Loop starts at 0,
    # Ends at 1 less than the last number
    # The loop increments 0, 1, 2, 3
    print("Square Right")
    for x in range(0, 4):
        # Print the loop counter
        print(x)
        gpg.led_off("right")
        gpg.drive_inches(
            distance,   # How far to drive in inches
            True        # Blocking, nothing else can happen while moving
        )
        gpg.led_on("right")
        # Turn right 90 degrees, positive number is right
        gpg.turn_degrees(90)
    # Turn both blinkers off
    gpg.led_off("right")
    gpg.led_off("left")


#------------------------- SQUARE LEFT ------------------------#
def square_left(distance):
    """
        Drive a left square based on the distance argument
    """
    print("Square Left")
    for x in range(0, 4):
        print(x)
        gpg.led_off("left")
        gpg.drive_inches(distance, True)
        gpg.led_on("left")
        # Turn left 90 degrees, - is left
        gpg.turn_degrees(-90)
    gpg.led_off("left")

```

```
#------------------------- GOPIGO WAGGLE ----------------------#
def waggle():
    """ Waggle back and forth """
    print("Waggle")
    for x in range(0, 4):
        print(x)
        gpg.led_on("left")
        gpg.turn_degrees(-10)
        gpg.led_off("left")
        gpg.led_on("right")
        gpg.turn_degrees(10)
        gpg.led_off("right")
    # Turn off both blinkers
    gpg.led_off("right")
    gpg.led_off("right")


def main():
    """ Main Program Entry Point """
    # Drive a 5" square turning left
    square_left(5)

    # Turn left to reverse the square
    print("Turn Left 90")
    gpg.turn_degrees(-90)

    # Drive a 5" square turning right
    square_right(5)

    print("Spin left.")
    gpg.spin_left()
    time.sleep(1)

    # Waggle back and forth
    waggle()

    print("Spin right.")
    gpg.spin_right()
    time.sleep(3)

    print("Stop!")
    gpg.stop()
    print("Done!")


# If a standalone program, call the main function
# Else, use as a module
if __name__ == '__main__':
    main()
```

Son of Driving School

You have seen this before in Intro to Robotics. Here it is again!

You will find several assignments from Intro to Robotics in the mBot folder in GitHub.

Minimum Requirements

1. Each movement will have its own function. This is demonstrated in the example program: square_left, square_right and waggle.

2. **DRY:** Don't Repeat Yourself (Reuse functions, build bigger functions from smaller functions.
3. Create a menu to choose which function you wish to perform.
4. This will be a single program that we will create in stages.

Stage 1

1. **Square** - your robot will trace the path of a square that is 1-foot square. It will start and end in the same place and the same orientation.
2. **Rectangle** - your robot will trace the path of a rectangle that is 1-foot x 2-foot. It will start and end in the same place and the same orientation.
3. **Sentry** - your robot will trace a 1-foot square around an object. Start the square one way, then turn around and go back the other way. Return to the beginning point and orientation.
4. **Retrace** - move in a 1-foot square forward, and then move in reverse to retrace that same square backwards to the beginning point and orientation. One solution would be to build a Reverse block that uses negative numbers for motor movement.
5. **ForwardReverse** - Move forward 12", turn 180°, move backwards 12" (which will be the same direction), turn 180° again, and then continue to move forward 12". The robot should move in one direction, but do part of the trip moving backwards.

Stage 2

1. **Octagon** - Move your robot in a 12" octagon. Each turn is a 45° angle. Start and end in the same place and the same orientation.
2. **Equilateral Triangle** – Move your robot in a 12" equilateral triangle. Start and end in the same place and the same orientation. An equilateral triangle has an inside angle of 60 degrees. Subtract that from 180 degrees to find out how far the robot should turn for each side.
3. **5-Point Star** – Teach your robot to trace a 5-point 12" star. Start and end at the same location and orientation. Look up the inside angle and subtract from 180 degrees.
4. **3-PointTurn** - Using 3 or more turns, teach your robot how to make a 3-point turn, like a regular car. You don't have to do curves, you can use straight angles if you wish.

Stage 3

1. **Circle** - your robot will trace the path of a circle that is 1 foot in diameter. It will start and end in the same location, and in the same orientation.
   HINT: Adjust the power of your left and right motors to create a left half circle block and right half circle block. Put those together to make your curved shapes.
2. **S-Shape** – your robot will trace two half-circles to create an S-shaped curve. Your robot will start and end in the same orientation, and the two half-circles will be the same size.
3. **Figure-8** - Move in a figure-8 shape.

Son of Random Numbers

Create and display random numbers on your GoPiGo terminal.

Son of Obstacle Avoidance

Another blast from the past!

Minimum Requirements

Start with a simple Obstacle Avoidance program. Add in other requirements as you work on this project.

- Use the shape of your mBot mBlock or Arduino code to help your GoPiGo learn autonomous obstacle avoidance.
- Use your servo to look right and left to gauge the distance.
- Use random numbers to change the obstacle avoidance.