piRover Builds with K2

piRover – Line Follower Input Rev 1.1

Overview:

This lesson extends the GPIO input functionality seen in the prior Pushbutton lesson. The line follower sensors (4) installed in the tracking module sense a line on the floor by emitting an infrared light toward the floor and measuring the amount of light reflected. When the black line (tape) is below the sensor, light reflection is minimal, and the sensor signal goes low.

Initially, you will consider the line follower inputs to complete a function table specifying the drive and LED indication required for each valid tracking module signal. You'll then create GPIO input code to sense and display each sensor's state. Finally, you will create a selection structure that outputs the drive status required for each valid state.

All output for this activity is provided in the terminal using print() function. Actual drive control needs to wait until the basics of the piRover drive are covered. See subsequent lessons.

Prerequisites:

Prior to beginning the instruction provided in this lesson you must have completed the following:

1. Pushbutton

Performance Outcomes:

1. Use the GPIO.input() function to read the logic levels from multiple GPIO inputs.
2. Determine drive and LED action from line tracker inputs.
3. Display line tracker status.
4. Determine appropriate drive and LED states for specific line tracker inputs.

Resources:

1. How to Make Ir Sensor Module

Materials:

1. piRover

Part 1 – Set Up

1. Prepare your workspace for this activity.
    a. Connect to your piRover using VNC.
    b. Access your piRover folder
    c. Create a **06.LineFollowerInput** directory
    d. Create a line_follower_input.py file.

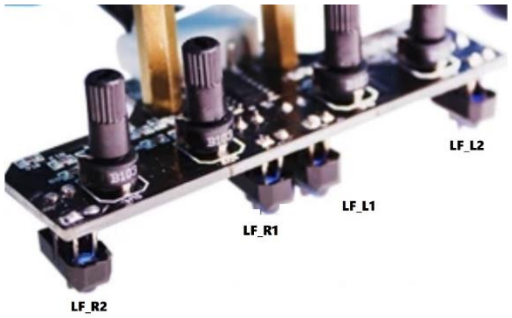
piRover Builds with K2

Part 1 – Investigate the Hardware

2. The line tracker uses infrared proximity detectors to sense the line. These sensors are made up of an infrared emitter diode and an infrared detector diode. See the How to Make Ir Sensor Module link for additional information on these devices and related circuity.

2. The line tracker module has four sensors. We'll use the following labels to identify.

   LF_L2 – Far left sensor
   LF_L1 – Near left sensor
   LF_R1 – Near right sensor
   LF_R2 – Far right sensor



3. In the prior Line Follower Preparation activity, you were asked to analyze each possible input combination and identify appropriate drive and LED outputs. The instructor's results are provided below. Were yours similar?

INPUTS				**OUTPUTS**		
LF_L2	**LF_L1**	**LF_R1**	**LF_R2**	**DRIVE STATE (Stop, Go, etc.)**	**LED COLOR**	**Comments (optional)**
0	0	0	0	N/A	N/A	All sensors are on
0	0	0	1	N/A	N/A	Both lefts are on but a right too!
0	0	1	0	N/A	N/A	Both lefts are on and a far right.
0	0	1	1	LEFT	Blue	L1 sensor is on so line is still close to center.
0	1	0	0	N/A	N/A	Both rights but a far left also.
0	1	0	1	N/A	N/A	Far left and near right
0	1	1	0	N/A	N/A	Far left and far right
0	1	1	1	HARD_LEFT	YELLOW	Far left is on. Turn toward line.
1	0	0	0	N/A	N/A	Both rights but near left too
1	0	0	1	GO	Green	Line is centered – Normal op
1	0	1	0	N/A	N/A	Near left and far right
1	0	1	1	LEFT	Blue	Slight left correction required
1	1	0	0	RIGHT	Blue	R1 sensor is on so line is still close to center
1	1	0	1	RIGHT	Blue	Slight right correction required
1	1	1	0	HARD_RIGHT	Yellow	Far right is on. Turn hard toward line.
1	1	1	1	STOP	RED	No sensor input. Line is lost.

Table 1 - Line Follower Function

4. Review the documentation in the Yahboom Expansion Board Manual to determine which GPIO pins are connected to the tracking module.

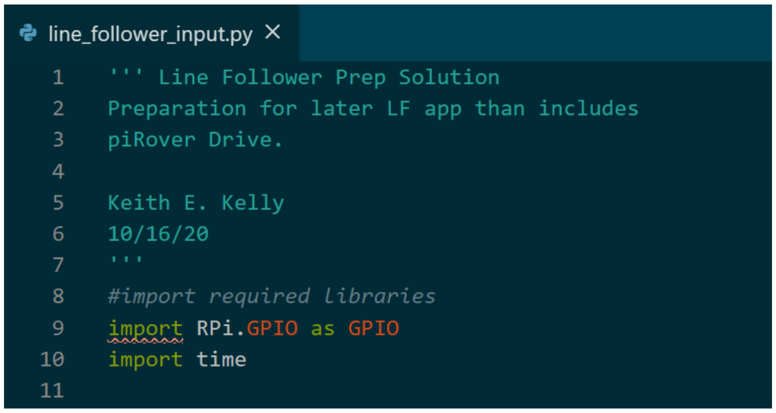
Table 2 - GPIO Pins

Sensors	Yahboom ID	Pin	GPIO
LF_L2	IN2	5	3
LF_L1	IN1	29	5
LF_R1	IN3	7	4
LF_R2	IN4	12	18

5. Your goal in this activity is to read the status of the line follower inputs and display in the terminal window. You will enable tracking by adding drive code in a later lesson. For now, you'll create the input and selection logic that specifies the action required by the drive.

Part 2 – Create the line_follower_input.py Code

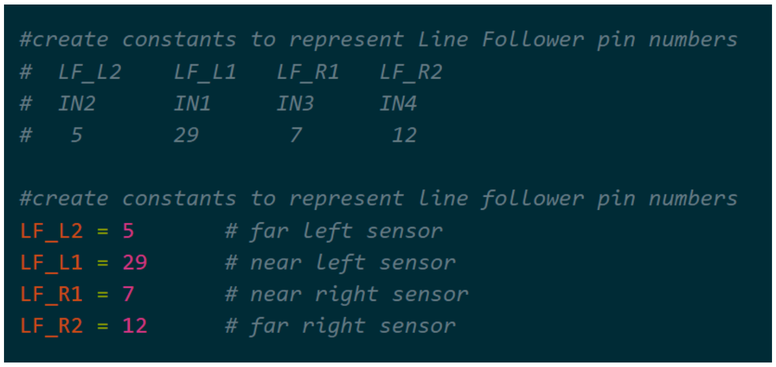
1. Open the line_follower_input.py file in the VS Code editor.
2. Enter a docstring header describing the Python file and import the required libraries.



line_follower_input.py

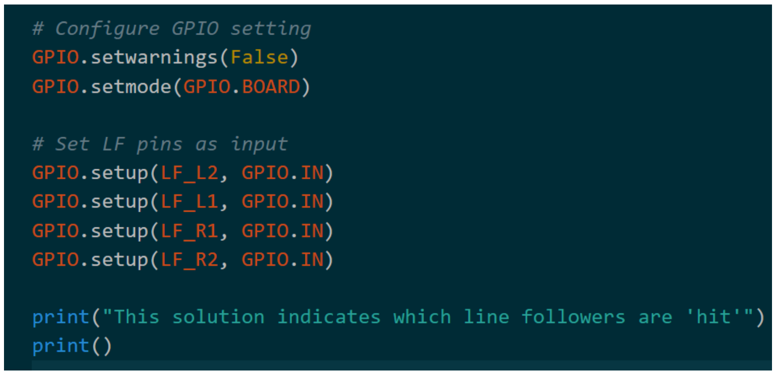
```
''' Line Follower Prep Solution
Preparation for later LF app than includes
piRover Drive.

Keith E. Kelly
10/16/20
'''
#import required libraries
import RPi.GPIO as GPIO
import time

```

3. Add constants to represent the required GPIO pin numbers.



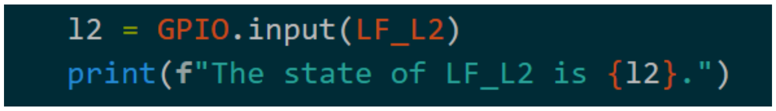
```
#create constants to represent Line Follower pin numbers
#   LF_L2     LF_L1    LF_R1    LF_R2
#   IN2       IN1      IN3      IN4
#    5        29        7        12

#create constants to represent line follower pin numbers
LF_L2 = 5        # far left sensor
LF_L1 = 29       # near left sensor
LF_R1 = 7        # near right sensor
LF_R2 = 12       # far right sensor
```

4. Configure the GPIO.



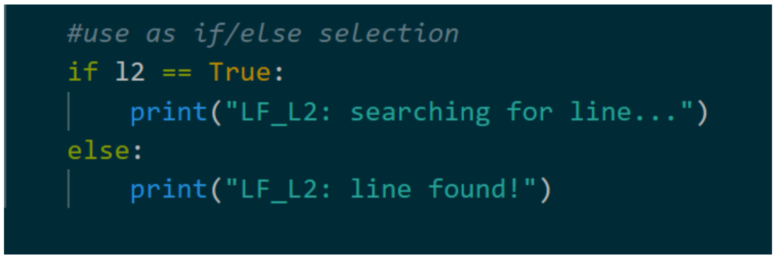
```python
# Configure GPIO setting
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

# Set LF pins as input
GPIO.setup(LF_L2, GPIO.IN)
GPIO.setup(LF_L1, GPIO.IN)
GPIO.setup(LF_R1, GPIO.IN)
GPIO.setup(LF_R2, GPIO.IN)

print("This solution indicates which line followers are 'hit'")
print()
```

5. Basic configuration is now complete. Create a while loop to continually read the line follower sensor inputs and print.
   a. Start with one sensor. Read the state of the sensor and print its status. Test your code.
   b. Add additional text to the output informing the user which sensor is being displayed. Try using the f-string format below.



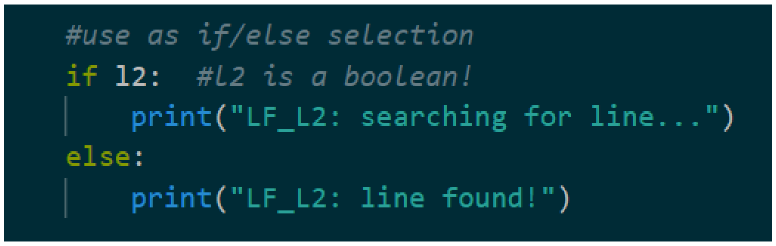
```python
l2 = GPIO.input(LF_L2)
print(f"The state of LF_L2 is {l2}.")
```

   c. Use an if/else structure providing more detailed output.



```python
#use as if/else selection
if l2 == True:
    print("LF_L2: searching for line...")
else:
    print("LF_L2: line found!")
```

   d. Modify the code above by recognizing that the l2 variable is a Boolean data type. The test for equal to True is not required.



```python
#use as if/else selection
if l2:  #l2 is a boolean!
    print("LF_L2: searching for line...")
else:
    print("LF_L2: line found!")
```

6. On your own, complete this initial version of the code that displays the status of each sensor to the user.

Part 3 – Investigate additional features of print()

7. Comment out the display code created in the section above. Do not delete. It is a required component of this activity.
8. Add the following print statement. Use the “IntelliSense” feature of VS Code for information on end=.

```
print(l2, end=" ")
print(l1, end=" ")
print(r1, end=" ")
print(r2, end="\n")
```

9. Comment out the four print statements and add the following single print statement. Use the “IntelliSense” feature of VS Code for information on sep=.

```
print(l2, l1, r1, r2, sep=" - ")
```

10. Comment out this print statement and add the following single print statement. See Python escape characters here.

```
print(l2, l1, r1, r2, sep="\t")
```

11. Add the following import statement at the top of your file. This enables access to the operating system library.

```
import os
```

12. Comment out prior print code and add the following. Test and note the output and format. Can you describe the function of each item?

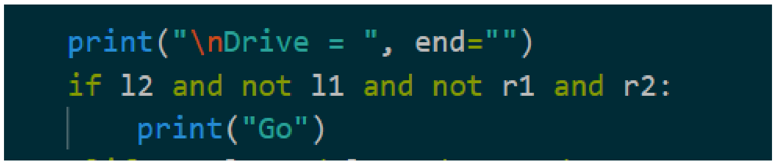
```
os.system("clear")
print("L2\tL1\tR1\tR2")
print(20 * "-")
print(f"{l2}\t{l1}\t{r1}\t{r2}")
```

13. Add a comment line above each line of code provided above that describes code output and format.

piRover Builds with K2

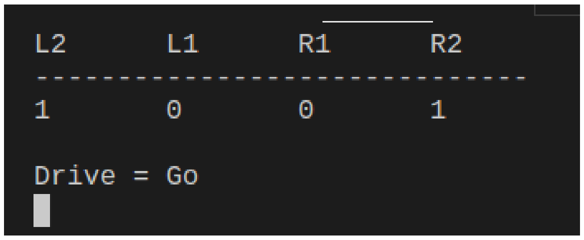
Part 4 – Line Follower Decisions

14. Add decision code showing drive actions required. The drive message and first conditional are provided below.



```
print("\nDrive = ", end="")
if l2 and not l1 and not r1 and r2:
    print("Go")
```

15. Continue the pattern shown to implement all drive conditions identified in Table 1
16. A sample of the final line_follower_input.py output is shown below.



```
L2        L1        R1        R2
------------------------------
1         0         0         1

Drive = Go
```

Assessment:

1. Submit to your line_follower_input.py to the Moodle site as directed.