BBC micro:bit
Sensing Light

Introduction

On the end of each of the fins is a tiny light sensor. With two of the three analog pins already in use for driving the motors, these two analog sensors are both connected to pin 2. There is a select pin, pin 16. You write a digital 0 to this pin to read the left light sensor, a 1 to read the right sensor.

Having two light sensors means that you can compare the two readings and use that difference in light levels to decide how to move the robot. You can also use average of the two readings to get a sense of the light level in a room.

The readings you get from light sensors vary a lot. You get a reading of 0 when it is completely dark to 1023 when there is bright light shining on the sensor. The background reading, when you are not shining a light on the sensor or covering it up, depends entirely on the lighting of the place where you are working.

Programming

This is a test program. You need to be using a PC with serial port drivers and something like Mu to view the REPL output. The function is used to get the readings for the left and right sensors, printing to the REPL window. Covering up the sensors and shining lights on them when running this program is useful.

```
from microbit import *

# select: 0 = left 1 = right
# dark 0, light 1023
def SenseLight(select):
    pin16.write_digital(select)
    return pin2.read_analog()


while True:
    print(SenseLight(0), SenseLight(1))
    sleep(50)
```

This is the simplest version of a light following program. It demonstrates the principle but needs a lot of optimisation. The function compares the two readings on the sensors and makes the robot head in the direction of the brighter light. You need to point a light source at the sensors to control the movement of the robot.

```
from microbit import *

# select: 0 = left 1 = right
# dark 0, light 1023
def SenseLight(select):
    pin16.write_digital(select)
    return pin2.read_analog()

def Drive(lft,rgt):
    pin8.write_digital(0)
    pin12.write_digital(0)
    if lft<0:
        pin8.write_digital(1)
        lft = 1023 + lft
    if rgt<0:
        rgt = 1023 + rgt
        pin12.write_digital(1)
    pin0.write_analog(lft)
    pin1.write_analog(rgt)

def HeadTowardsLight():
    lft = SenseLight(0)
    rgt = SenseLight(1)
    if lft>rgt:
        # head left
        Drive(100,400)
    elif rgt>lft:
        # head right
        Drive(400,100)
    else:
        #straight on
        Drive(400,400)
    sleep(20)

while True:
    HeadTowardsLight()
```

Challenges

1. Have the robot wake up when it detects a big change in light on either sensor. Start with a sleep statement to give you time to turn out the lights after setting up the robot. You will want 1000 for each second you need. Then, assign a reading from the sensors to a variable. Use a while loop to repeatedly take readings from the sensors while they remain below a threshold. After this, make the robot do something loud and bright.
2. The HeadTowardsLight function is far from perfect. With a little tweaking, you can get better responses from the torch remote control if you have a small dead zone, a tolerance that allows you

BBC Microbit

Collapse All Expand All

+ Block Editor - The Basics
+ Block Editor - Components
+ Kodu - micro:bit Worlds
+ JavaScript Blocks
+ JavaScript Blocks - Exercises
+ Blocks - Bit:Bot
+ Blocks - Bit:Commander
+ MicroPython - Starting Off
+ MicroPython - Examples
+ MicroPython - Components
+ MicroPython - Breakout Boards
+ MicroPython - Exercises
+ MicroPython - Pi Accessories
- MicroPython - Bit:Bot
  - Meet The Bit:Bot
  - Driving The Motors
  - Beeping The Horn
  - Lighting The Way
  - Following Lines
  - Sensing Light
  - Radio Control
+ MicroPython - Bit:Commander
+ MicroPython - Projects
+ MicroPython - Visual Basic
+ Other - Odds & Ends

to move in a straight line more easily. In the function, you could make the first clause of the if statement check if abs(lft - rgt) is less than the size of the dead zone and move forwards if it is. Try a number like 50 to start with and vary until you see if the effect work with your light source and conditions. You can also think more carefully about how much you want to turn the vehicle. You can experiment with the kinds of turn that you want the robot to make.

3. What effect does the light from the neopixels have on the light sensors?
4. If you can get the robot to travel to and from a point to all points on the compass, a set distance away, you can make a room scanning robot. Have a robot try each of the compass points and go to the one with the brightest light. Repeat the process again. An alternative is to to go to each one in sequence and take the first one that is brighter than the starting point before repeating the process.
5. If you have a light source that is easy to flash, then program the robot to respond to specific sequences of light. For example, a flash of light, followed by a pause between 500 - 1000 milliseconds, another flash, another pause, another flash. This makes the robot lights come one. Do the same again to switch them off. Work out how to tell the difference between long and short flashes of light. Now you have the tools to receive Morse code. You need to allow some tolerance, for a human user and experiment with the time frames so that things are repeatable to you. Find out how to make a dictionary using the .- characters as keys and letters as the values. If you can encode a sequence of flashes as a string of dots and dashes, you can convert them to letters. If you crack the Morse code, then use it to trigger some intersting behaviours from the robot, some light, sound etc. You could, for example, use Morse code to change the colour displayed on the neopixels or have the robot drive in a circle, flash its lights a bit, wiggle about. It's about the slowest and trickiest remote control ever but a great idea nonetheless.