BBC micro:bit Radio Control

Introduction

Let's be honest. When you think about a robot car, you are wanting to be able to control it remotely. Fun as it is laboriously working out code to direct a robot around a complex route, it does seem easier to be able to respond in real time when things aren't quite what you expect.

With only one GPIO spare on the Bit:Bot and no more analog pins, you'd think that it might be difficult. Fortunately, the radio library for MicroPython is pretty well designed. A second micro:bit can be used as a remote control for the robot car.

Programming - Sending

Here is my code for the remote control. I'm using the accelerometer to go backwards and forwards by tilting the micro:bit and using the two buttons to make the robot turn left and right. Some arrows are used on the LED matrix to help with debugging.

```
from microbit import *
import radio

chnl = 10
radio.config(channel=chnl)
radio.on()


while True:
    y = accelerometer.get_y()
    a = button_a.is_pressed()
    b = button_b.is_pressed()
    if  a and y<-300:
        # forwards left
        display.show(Image.ARROW_NW)
        radio.send("NW")
    elif a and y>300:
        # backwards left
        display.show(Image.ARROW_SW)
        radio.send("SW")
    elif b and y<-300:
        # forwards right
        display.show(Image.ARROW_NE)
        radio.send("NE")
    elif b and y>300:
        # backwards right
        display.show(Image.ARROW_SE)
        radio.send("SE")
    elif y>300:
        #backwards
        display.show(Image.ARROW_S)
        radio.send("S")
    elif y<-300:
        # forwards
        display.show(Image.ARROW_N)
        radio.send("N")
    sleep(20)
```

Depending on the input, or combinations of input, a compass direction is sent to indicate the direction in which to drive the robot. A channel number is used. Radio isn't a direct one-to-one form of communication. If there are other people using the micro:bit radio in your vicinity, the receiving micro:bit is going to pick up and respond to those signals. There are channels numbered from 0 - 100. Our receiving micro:bit will only receive the messages that are sent on the same channel. The default value is 7. Change from that and no one else's radio can interfere with yours.

Programming - Receiving

Here is the script that receives the radio signal and drives the robot in the correct direction.

```
from microbit import *
import radio

chnl = 10
radio.config(channel=chnl)
radio.on()

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)

while True:
```

```
    s = radio.receive()
    if s is not None:
        if s=="N":
            Drive(800,800)
        elif s=="S":
            Drive(-800,-800)
        elif s=="NE":
            Drive(800,200)
        elif s=="NW":
            Drive(200,800)
        elif s=="SE":
            Drive(-800,-200)
        elif s== "SW":
            Drive(-200,-800)
    else:
        Drive(0,0)
    sleep(20)
```

You might wish to experiment a little with this to get the best performance. My aim was to get a working version. This isn't bad and gave me precise enough control of the robot to drive it around within an area where you'd expect to receive the signal. You can configure the radio to use more power (see the MicroPython documentation) for the radio but I found that default values were more than enough for the distance I'd normally expect. Control was a little jerky on some manouevres but the robot was pretty controllable.

Challenges

1. The first challenge is to improve the range of movements and the amount of control you feel over the robot. You could use the accelerometer for more than just backwards and forwards. That would free up the buttons for more functionality. I like to be able control the buzzer for beeping and be able to flash lights when I feel like it. My feline test subjects prefer it that way too.
2. The example on this page doesn't make full use of the GPIO on the micro:bit that does the sending. You could go for a load of buttons in a much more fancy controller. Use a keypad or an input shift register and you can have full mission control. That would give you the scope to remotely trigger the robot to perform sequences of pre-designed actions.
3. The radio is a two way thing. The robot can send feedback. The robot could send messages to the controller to confirm the movements that it is doing and they could be displayed on the controller matrix.
4. With a second micro:bit, you can hookup an LCD screen or use the matrix to display information from the light sensors. With a specially designed course, you could use the line sensors to read information from the track about the position of the robot. Could you work out a protocol and some programs for being able to know where the robot is on the track just from the data it sends you (not looking at the robot)? Think about how you could set up a system of markings that you could recognize with the line sensors. Work out how to make this is compact as possible, taking up as little space as possible on the track/course.
5. The Neopixels on the robot are pretty cool in themselves. The car makes a nice mood lamp. With a second micro:bit controlling the colour and or the effects on the lights, you can make something interesting withouth even bothering the motors. You could set up a simple line for the robot to follow slowly - something it could follow without having to make too many course corrections. Make it make back and forward using the line to ensure you get to the correct spot. Either that or place find a transparent or translucent container and stick the robot in that with no moving aboout involved. You could use the buttons to trigger different colours and animations of the neopixels. Light show.