C - Smoother Line Following

Learn about analogue inputs to make an even more sophisticated line following robot, that will smoothly follow any path.

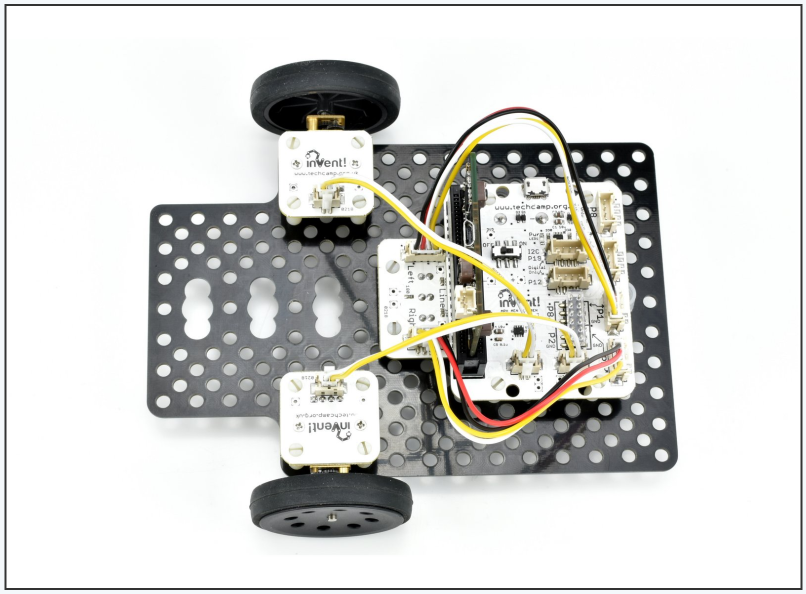
INTRODUCTION

Learn about analogue inputs to make an even more sophisticated line following robot, that will smoothly follow any path.

Step 1

Setup Your Robot

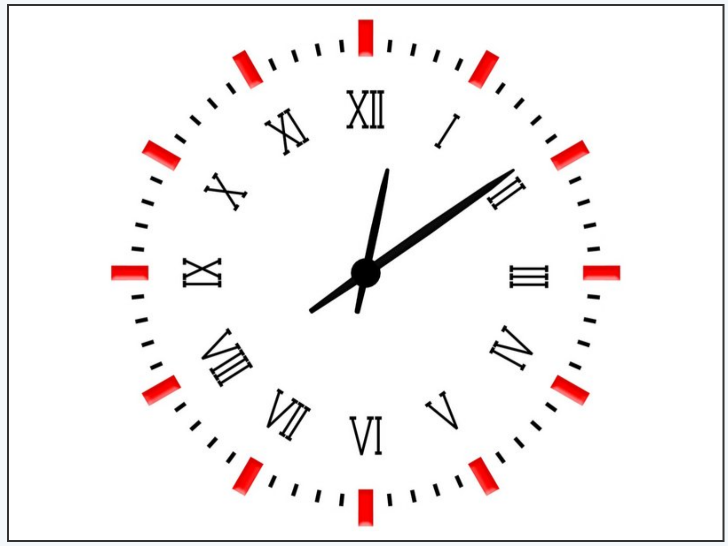
- We just need the **line sensor** for now - make sure your robot is setup like the picture.



This document was generated on 2021-12-25 08:56:20 AM (MST).

Step 2

Analogue and Digital



- To make a **smoother**, better line follower, we need to use the line sensor in **analogue mode.**
- So far, we have been using it as a **digital** sensor - it can only be **ON or OFF (1 or 0).**
- Analogue inputs (and outputs) can have **any value** - think about the difference between a digital and an analogue clock
- A digital clock must display a **whole number** of minutes
- But on an analogue clock, the minute hand can be **anywhere** - even halfway between two minutes!

Step 3

Analogue Line Sensor

- **Build** the simple test program in the picture.
- Program your robot, and **keep it plugged in.**
- Try moving the robot **slowly** from one side of the line to the other, whilst watching the speeds of the motors.
- See how the change **gradually** as you approach the line?

```
    p[6]=(p[5]+p[3])/2;p[7]=(p[4]+p[2]).
    while(p0()>p[7]and p1()>p[6]):d(0 i
def digital_read_line(s): return 1 if (
def analog_read_line(s): v=p0()if s==0 
# Invent! Code End
# Start your code below here!

calibrate_line_sensors()

while True:
    left=analog_read_line(1)
    drive_motor(0,left)
```

Step 4

2 Analogue Sensors

- We can use this **gradual** change to **smoothly** change the amount the robot turns as it get **further from the line!**
- Add **two new variables** to the program inside the while True: loop, called **l** and **r** (left and right).
- Let **l** = the analogue value of the **left sensor (1)**, and **r** = the analogue value of the **right sensor (0).**

ⓘ The analog_read_line function returns **100** if the sensor is completely **off** the line (on white), and **0** if the sensor is completely **on** the line (black).

```
    while(p0()>p[7]and p1()>p[6]):d(0 i
def digital_read_line(s): return 1 if (
def analog_read_line(s): v=p0()if s==0 
# Invent! Code End
# Start your code below here!

calibrate_line_sensors()

while True:
    l=analog_read_line(1)
    r=analog_read_line(0)
```

This document was generated on 2021-12-25 08:56:20 AM (MST).

Step 5

How much to turn?

- The larger the **difference** between l and r, the further the robot is from the line so the **more** we need to turn.
- For example, if both sensors are on the line, we **don't need to turn at all** and l and r will have the **same value.**
- Add a new variable called **turn.**
- **After** getting the values of l and r, set turn equal to the **difference** between **l and r.**

```
def digital_read_line(s): return 1 if (
def analog_read_line(s): v=p0()if s==0
# Invent! Code End
# Start your code below here!

calibrate_line_sensors()

while True:
    l=analog_read_line(1)
    r=analog_read_line(0)

    turn=l-r
```

Step 6

Turning

- Then, **add two lines** of code like the picture, to set the speeds of the motors **using the turn variable.**
- Do you **understand** how the code works? (hint: turn is **positive** when we need to turn **right**, and **negative** when we need to turn **left**)

```
def analog_read_line(s): v=p0()if s==0
# Invent! Code End
# Start your code below here!

calibrate_line_sensors()

while True:
    l=analog_read_line(1)
    r=analog_read_line(0)

    turn=l-r

    drive_motor(1,50+turn)
    drive_motor(2,50-turn)
```

This document was generated on 2021-12-25 08:56:20 AM (MST).

Step 7

Maximum Speed

- You might have noticed that while the new program is **smooth**, it isn't as **fast** as the old two sensor **digital** program - it might also struggle with the **tighter turns.**
- To make it faster, we need to make sure 1 wheel is always going **100% forwards**, and then change the speed of the **other wheel only** based on how large **turn** is to follow the line.
- **Change your program** so it looks like the picture - this will make sure 1 wheel is always going at 100%.
- To make this work for tight turns, we need to **multiply** the turn variable to it has a bigger effect. Try it out with 3 to start with - you might need to **adjust** this depending on your exact robot setup, and how tight the turns are on the line.
- **Be sure to test it properly** - try adjusting things until your program is **100% reliable.**

```
# Invent! Code End
# Start your code below here!

calibrate_line_sensors()

while True:
    l=analog_read_line(1)
    r=analog_read_line(0)

    turn=l-r

    drive_motor(1,100+(turn*3))
    drive_motor(2,100-(turn*3))
```

Step 8

Proportional Sparkles

- If you're feeling really advanced, add the **Sparkle module** back in and set the colours of the LEDs **proportionally** based on how far away from the line the robot is!
- Your robot can also **get lost** and now has no way of finding the line again - try and **add the code you wrote previously** back in so the robot can't get lost, or at least **stops** if it loses the line completely.

Extension Challenge!

This document was generated on 2021-12-25 08:56:20 AM (MST).