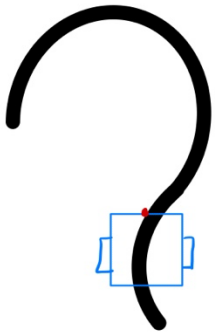
Why we used 2 sensors?

With one sensor:

- in the middle of the robot
- pointing down
- placed between the left side of the light and the white surface (Let’s call it the 'grey part')



First thing to take in consideration:

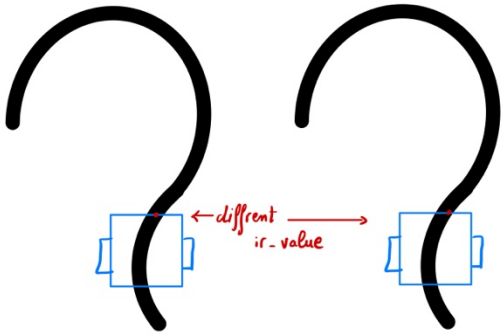
Our line is highly curved, basically there is no straight parts + It's not curved in one direction.

The robot will need to turn either to left or right mostly for each step it takes.

⇨ 1st Problem: too many turns.

Second thing:

With one sensor for E-Puck in Webots, IR sensor is able to detect the line.
Although the line is in black (one color), the ir_value that we get when the sensor is on line (on black) in position 1 is different from the ir_value that we get when the sensor is on line (on black) but this time in position 2. Sometimes even if the position on the line is the same, the returned value ir_value when starting the robot for the first time is different when starting it for the 2nd time.



After some debugging, we've noticed that when the sensor:

- on white -> returned ir_value < 6 or >= 1000

- as soon as it touches the black line the value is not 6 but > 6. It can be 27 or it can be 8 or it can be 6.2 AND < 1000.

In the hope to catch the 'grey part''s correct value, we gave the robot a very low speed so that it can recognize that part, but the results always variates between the given interval.

Okay, let's take these intervals in consideration and try to move our robot:

If we have a curved line to the left, yes, we can use one sensor:

Logic

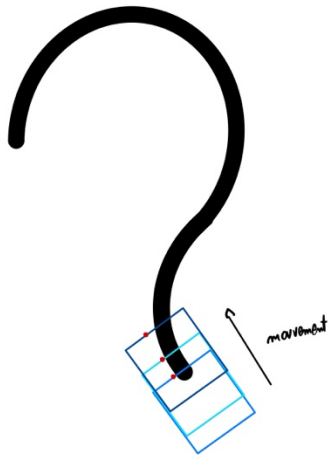
- on white -> go straight forward
- as soon as you touch the line -> turn left

Code

```
while robot.step(time_step) != -1:
        ir_value = left_ir.getValue()
        print("ir_value: {}".format(ir_value))

        left_speed = max_speed
        right_speed = max_speed

        if(ir_value > 6 and ir_value < 1000):
            print(" *****on black")
            #turn left
            left_speed = -max_speed
        elif((ir_value < 6 or ir_value >= 1000) ):
            print("*****on white")
  #no need to turn

left_motor.setVelocity(left_speed)
right_motor.setVelocity(right_speed)
```

No magic here no PID. I am just trying to explain what will happen if we use one sensor. No error is calculated (YET)= no Kp.

But if our curve at some point goes to the right side: the robot will keep going forward because it won't detect the line at all.



This logic
if ir_value > x turn right
if ir_value < x turn left
if ir_value == x keep going forward
won't work in Webots with one sensor.

Our robot will only interfere when the line is curved to the left. It's like if it has a blind right eye. If the line is curved to the right, it’s like if there is no line at all (just white space).

We didn't stop at this point and we went a little bit further:

I came up with the idea that, we'll create an extra parameter to know if the robot stayed on the white side for a long time. Please keep in mind that our line is highly curved, so our robot normally won't stay for long on the white side. There is always a turn it should make.
Logic

- if after a specific i number of steps, you didn't make a turn to the left (you didn't find any line), be aware that something went wrong and start turning to the right in order to find the line.

After some debugging:
Code

```
while robot.step(time_step) != -1:
        ir_value = left_ir.getValue()

        print("ir_value: {}".format(ir_value))

        left_speed = max_speed * 0.1
        right_speed = max_speed * 0.1

        if(ir_value > 6 and ir_value < 1000):
            print(" *****in black => i: {}".format(i))
            #turn left
            print(" -------------------------Go left")
            left_speed = -max_speed *  0.00001
            i=0

        elif(ir_value < 6 or ir_value >= 1000 ):
            i=i+1
            print(" *****in white => i: {}".format(i))
            if(i>25):
            #turn right
                print(" **********************Go right")
                right_speed = -max_speed *  0.00001

        left_motor.setVelocity(left_speed)
        right_motor.setVelocity(right_speed)
```

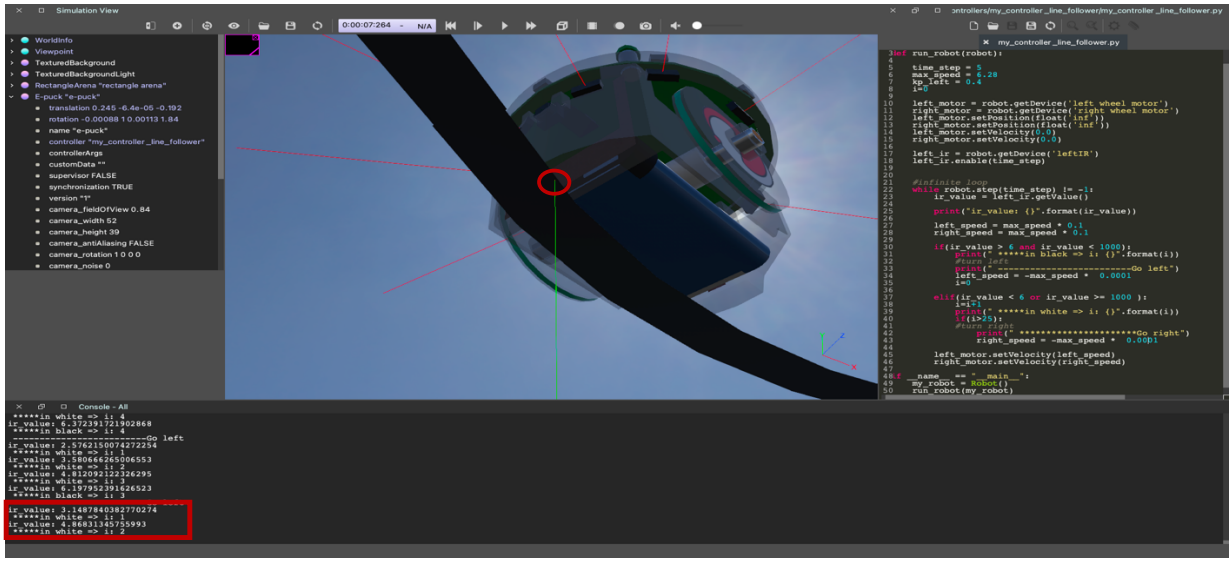
And you can find the corresponding video in this link:

https://drive.google.com/file/d/1X6rae2_uM__zcmPIvpUWvx-pzpex0Hno/view?usp=sharing

The problem with this is that we had to lower the speed of the robot to make the right turn smoother which made our robot too slow.

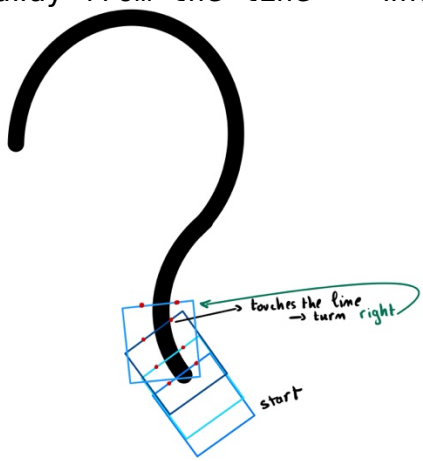
We weren't sure if the implementation of the 'i numbers of steps' is the correct way and another con here was the low speed.

- ⇨ But if we make the speed higher, we'll face another problem, which is:
  - Even if the sensor is on the line (normally the value should be > 6), as you can see below, our robot didn't detect that. Which leads us to another limitation of Webots.



With 2 sensors, it's simpler:

If the line, whose width (2cm) is smaller than the distance between the sensors (3cm), is between the two sensors and both of the sensors don't touch the line, our robot will keep going straight forward. So, in this case (2 sensors case) the robot won't be turning with each step. It will turn only when it drifted too far away from the line => when one of our sensors touches the line.



Why at least 2 sensors?

Our idea was to minimize the number of turns to give the robot (form our point of view) a smoother movement + make the robot move faster and detect the lines correctly. Solution: the use of at least 2 sensors.

Why did we use DistanceSensor?

Concerning the use of DistanceSensor. It's one of the other limitations of the Webots, after some research, the only sensor that can detect our line in Webots is the DistanceSensor of type IR. We have tested other sensors but they weren't able to detect the line.

I want also to add something: the provided photo about how IR sensor works was a general picture. In the documentation of 'E-Puck in Webots', it's written that:

> *'The "e-puck_line.wbt" world especially examplifies the use of ground sensors.... These sensors are actually simple infra-red sensors which allow the e-puck robot to see the color level of the ground. ... . This is particularly useful for implementing line following behaviors.*

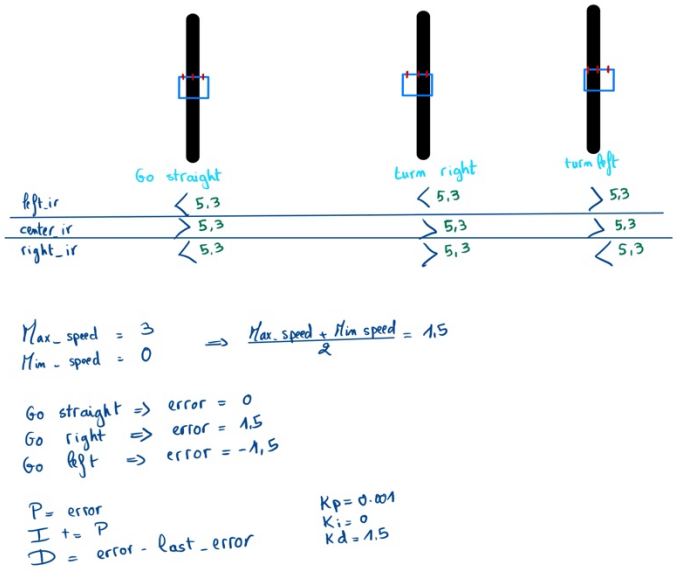
PID implementation

After calculating the PID term with the use of E-Puck with 2 IR-Sensors, the result wasn't 100% correct. Normally the PID and the P should be 0 after adjusting the robot (keep it in balance and make the right moves).

With 3 Sensors:

left_ir, center_ir and right_ir (the 3 red dots). After some calibration:

- Logic



- code

```
from controller import Robot

def run_robot(robot):

    time_step = 32
    max_speed = 3
    my_value=5.3

    kp=0.001
    ki=0
    kd=1.5

    P=0
    I=0
    D=0

    error= 0
    last_error= 0


    left_motor = robot.getDevice('left wheel motor')
    right_motor = robot.getDevice('right wheel motor')
    left_motor.setPosition(float('inf'))
    right_motor.setPosition(float('inf'))
    left_motor.setVelocity(0.0)
    right_motor.setVelocity(0.0)

    left_ir = robot.getDevice('leftIR')
    left_ir.enable(time_step)

    right_ir = robot.getDevice('rightIR')
```

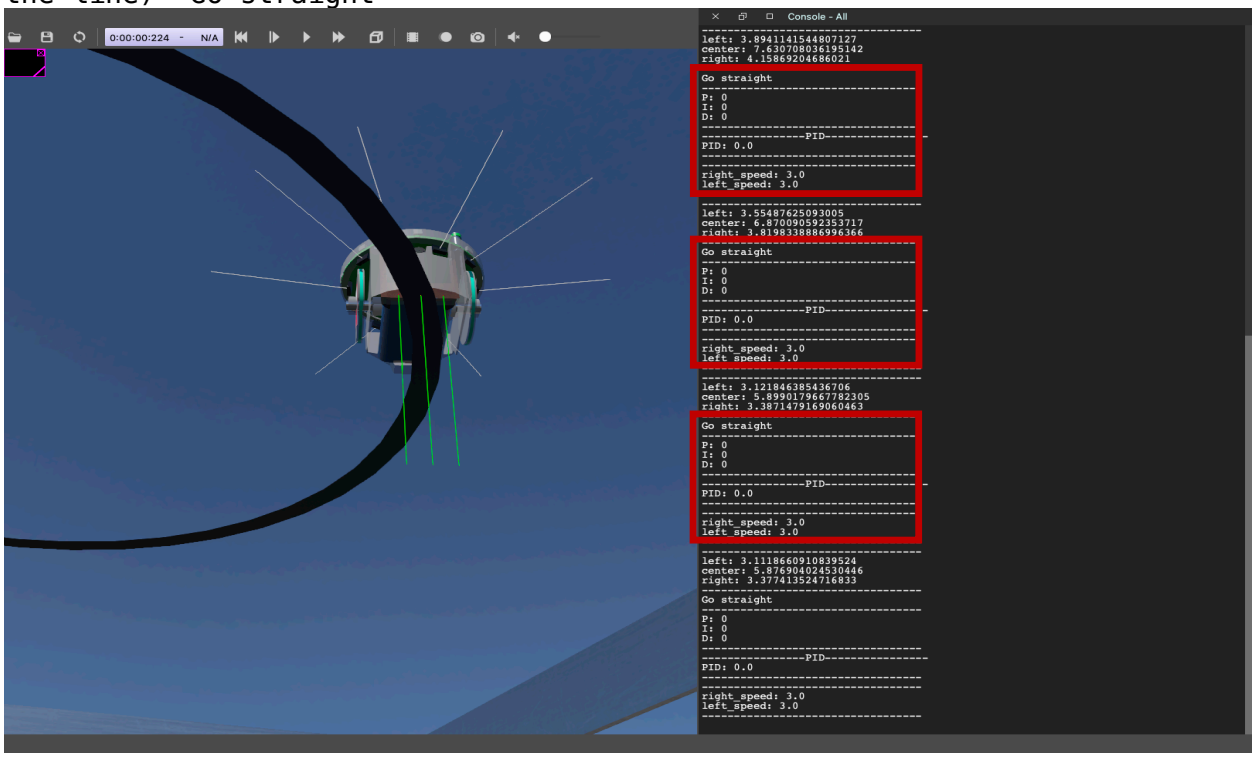
```
    right_ir.enable(time_step)

    center_ir = robot.getDevice('centerIR')
    center_ir.enable(time_step)



    #infinite loop
    while robot.step(time_step) != -1:
        left_ir_value = left_ir.getValue()
        right_ir_value = right_ir.getValue()
        center_ir_value = center_ir.getValue()


        print("-----------------------------------")
        print("left: {}".format(left_ir_value))
        print("center: {} ".format(center_ir_value))
        print("right: {} ".format(right_ir_value))
        print("-----------------------------------")

        left_speed = max_speed
        right_speed = max_speed

        #left: out center: in righet: out
        if(left_ir_value < my_value and center_ir_value > my_value and right_ir_value <
my_value):
            print("Go straight")
            error = 0
            P=error
            I= error + I
            D = error - last_error
            print("-----------------------------------")
            print("P: {}".format(P))
            print("I: {} ".format(I))
            print("D: {} ".format(D))
            print("-----------------------------------")
            PID =(kp*P) +(ki*I) +(kd*D)
            print("----------------PID----------------")
            print("PID: {}".format(PID))
            print("-----------------------------------")
            last_error = error
            left_speed= max_speed - PID
            right_speed= max_speed + PID
            print("-----------------------------------")
            print("right_speed: {}".format(right_speed))
            print("left_speed: {}".format(left_speed))
            print("-----------------------------------")



        #left: in center: in righet: out
```

```
        elif(left_ir_value > my_value and center_ir_value > my_value and right_ir_value <
my_value) :
            print("GO left")
            error = -1.5
            P=error
            I= error + I
            D = error - last_error
            print("-----------------------------------")
            print("P: {}".format(P))
            print("I: {} ".format(I))
            print("D: {} ".format(D))
            print("-----------------------------------")
            PID =(kp*P) +(ki*I) +(kd*D)
            print("----------------PID----------------")
            print("PID: {}".format(PID))
            print("-----------------------------------")
            last_error = error
            right_speed = max_speed - PID
            left_speed = max_speed + PID
            print("-----------------------------------")
            print("right_speed: {}".format(right_speed))
            print("left_speed: {}".format(left_speed))
            print("-----------------------------------")

        #left: out center: in righet: in
        elif(left_ir_value < my_value and center_ir_value > my_value and right_ir_value >
my_value) :
            print("Go right")
            error = 1.5
            P = error
            I= error + I
            D = error - last_error
            print("-----------------------------------")
            print("P: {}".format(P))
            print("I: {} ".format(I))
            print("D: {} ".format(D))
            print("-----------------------------------")
            PID =(kp*P) +(ki*I) +(kd*D)
            print("----------------PID----------------")
            print("PID: {}".format(PID))
            print("-----------------------------------")
            last_error = error
            right_speed = max_speed - PID
            left_speed = max_speed + PID
            print("-----------------------------------")
            print("right_speed: {}".format(right_speed))
            print("left_speed: {}".format(left_speed))
            print("-----------------------------------")
```

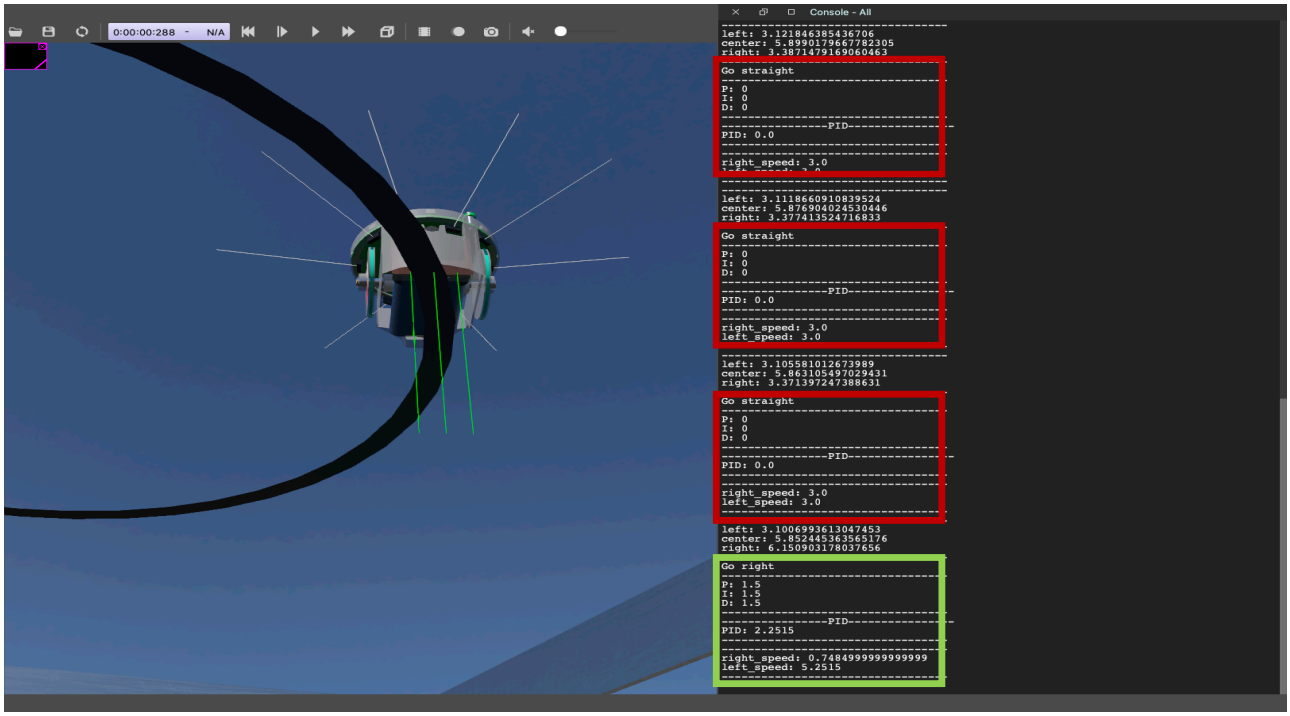
```
        left_motor.setVelocity(left_speed)
        right_motor.setVelocity(right_speed)

if __name__ == "__main__":
    my_robot = Robot()
    run_robot(my_robot)
```

- Output

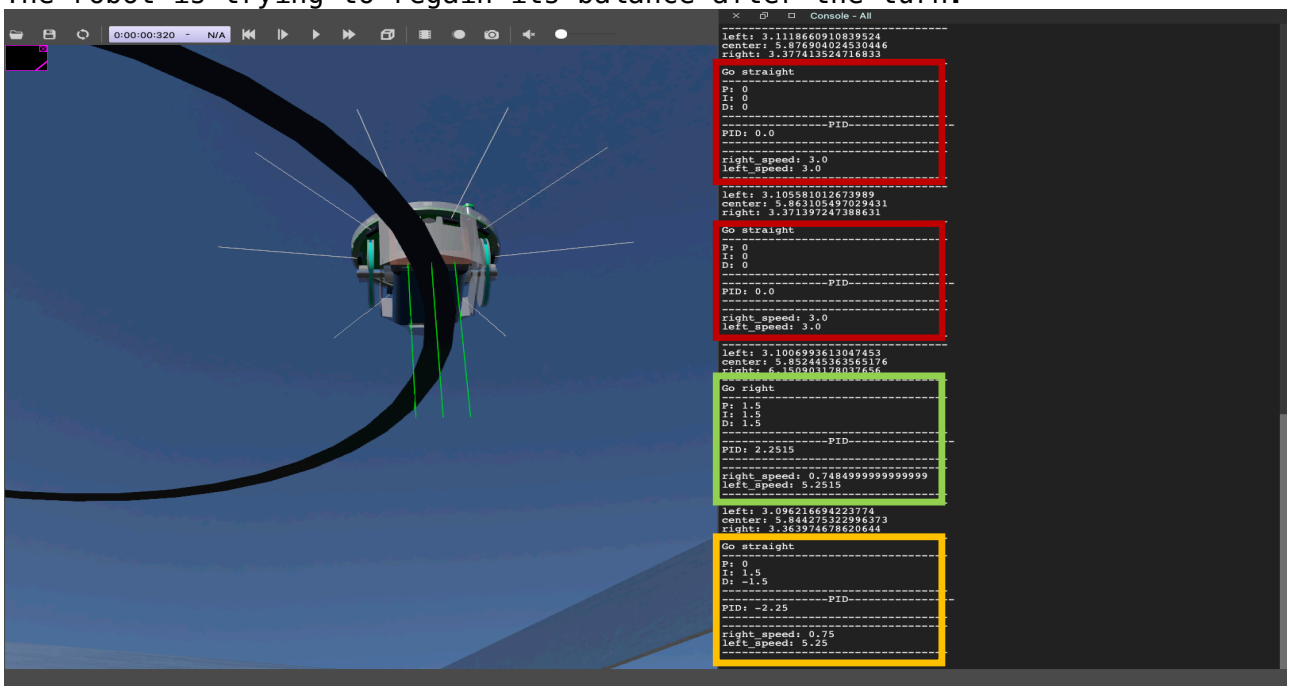
The robot is at this point in balance (only the center_ir is touching the line) =Go straight



As soon as the right_ir touches the line = Go right



The robot is trying to regain its balance after the turn.



The next step= Go straight