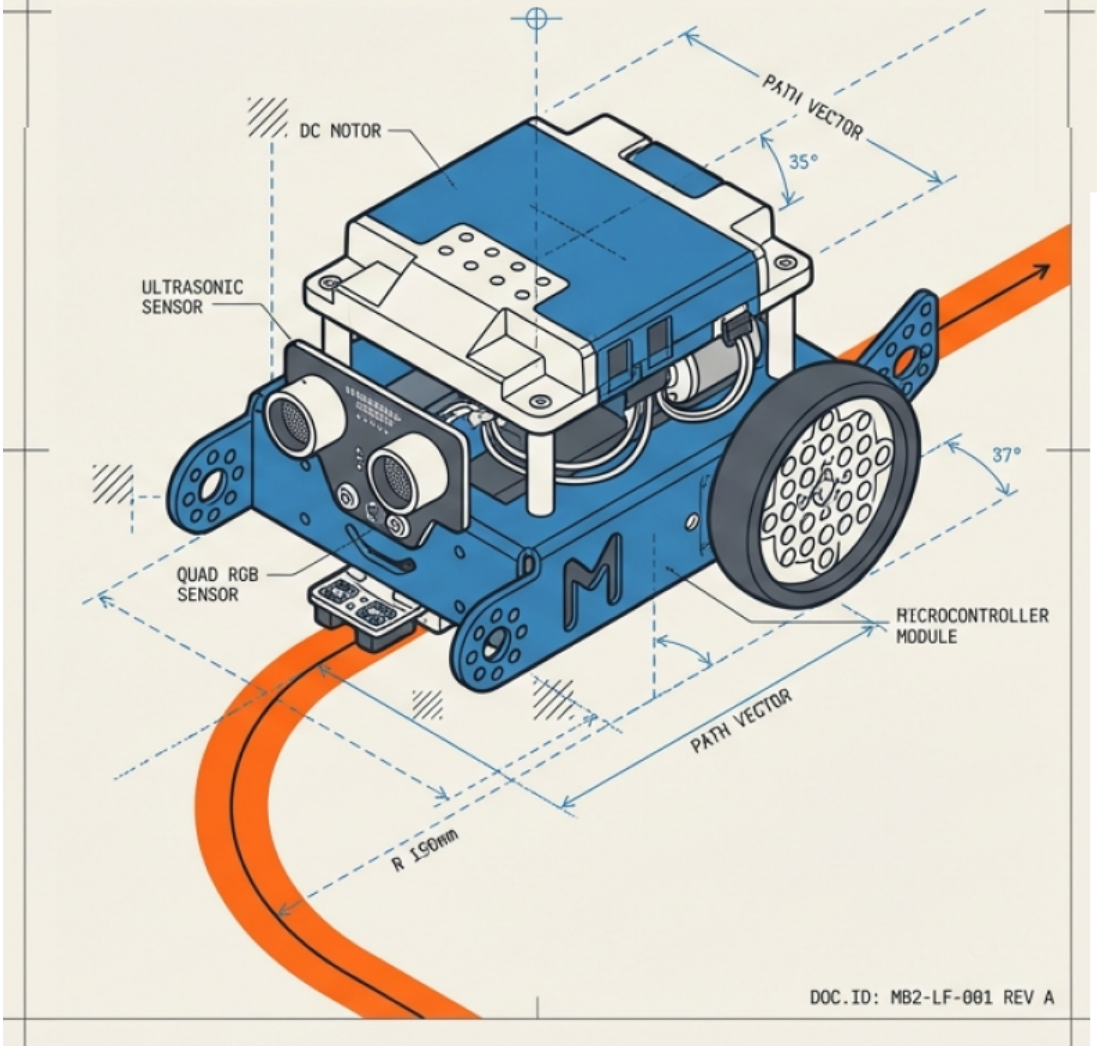
Mastering Line Following

From basic binary logic to mathematical proportional control for the mBot2.



1 Demo

The best way to see how the robot follows a line is to run built-in program mBot2 demo 2, which comes pre-loaded on every mBot2.

How to launch built-in programs:

1. Turn on the robot.
2. Enter the menu: if you are not in the main menu, press the Home button below the joystick.
3. Move the joystick down to *Switch Program* and confirm by pressing the joystick.
4. Place the robot on the line.
5. Select `mBot2 demo 2` and press button B to start.

If the robot does not follow the line well, the sensor needs to be calibrated.

2 Calibration

The fastest way to calibrate is without a computer. You will need a flat surface with normal room lighting (the same lighting you will use during the session) and white paper with a black line, e.g. the track poster included with the mBot2 kit.

Steps:

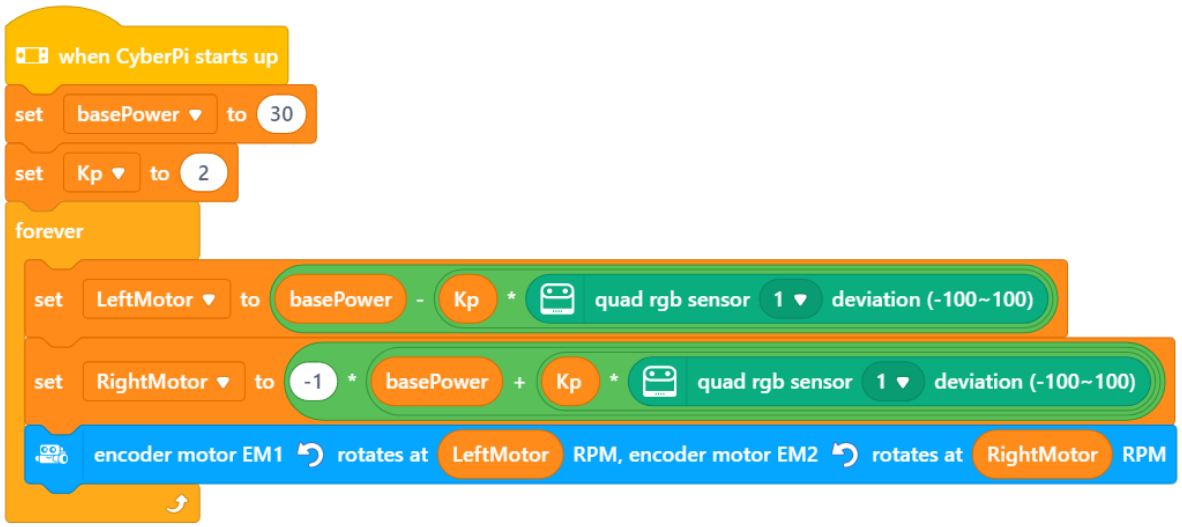
1. Place the robot on the **white area** of the paper.
2. Check that the sensor is **12–13 mm** above the surface.
3. **Double-press** the button on the sensor.
4. The LEDs start blinking in different colours – calibration is in progress.
5. Slowly slide the robot so the sensor **passes over the black line and back** to white.
6. When the LEDs stop blinking, calibration is complete.

The sensor is now calibrated to distinguish the line from the background.

3 Line Follower Program

Below is the line-following program that measures the robot's deviation from the centre of the line. The program is shown both in block-based and Python form.

mBlock – P-controller



Python – P-controller

```
import cyberpi, mbuild

basePower = 30
Kp = 0.5
LeftMotor = 0
RightMotor = 0

while True:
    LeftMotor = (basePower - Kp * mbuild.quad_rgb_sensor.get_offset_track(1))
    RightMotor = -1 * ((basePower + Kp * mbuild.quad_rgb_sensor.get_offset_track
        (1)))
    mbot2.drive_speed(LeftMotor, RightMotor)
```

If you want to truly understand the algorithm and use it in your own project, we recommend working through all the following worksheets in order.

Note

Tip: repeatedly plugging and unplugging the USB cable can get tedious. We recommend using a Bluetooth dongle for wireless communication. Makeblock offers a dedicated USB 2.0 Bluetooth Adapter for PC Connectivity.

Name: _____ **Class:** _____ **Date:** _____

4 Sensor Description

The Quad RGB sensor is a four-channel sensor for colour and line detection, mounted on the mBot2 robot. Its four channels are labelled L1, L2, R1 and R2 (L = left side, R = right side). The sensor measures reflected light intensity and internally compares the combination of values against a set of preset colours. It can detect 6 colours plus black and white, enabling mBot2 to follow a line on the floor and react to coloured markers. The sensor is located at the front of the robot. On the underside you can see the four sensing elements. The small button on the top of the sensor is used for calibration.

When programming mBot2 you will use this sensor to track colours on the track – for example, to stop at red or turn left at green.

The Quad RGB sensor has **4 sensing elements**, labelled as shown below (top view):

```
port 4   port 3     port 2   port 1
 (L2)     (L1)       (R1)     (R2)
  ●        ●          ●        ●
```

The sensor also has a **button** and **coloured fill-light LEDs** that illuminate the surface with visible light.

4.1. Physical Layout

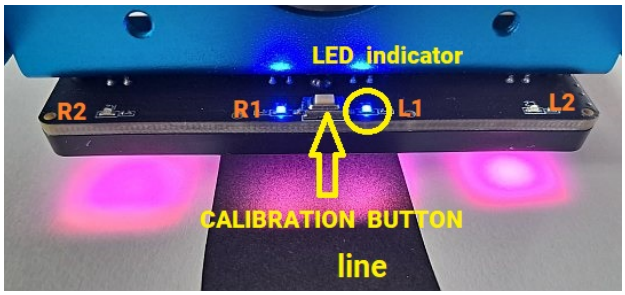


Figure 1. Sensor on the robot; blue LED indicates line detection

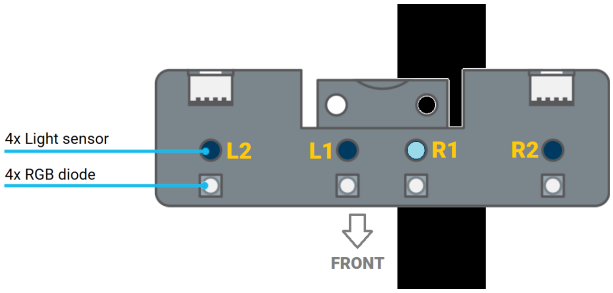


Figure 2. Bottom view – sensor labels

5 Why Calibration?

This RGB sensor measures reflected light intensity. The problem is that **black** and **white** are not strictly fixed values – in practice they depend on:

- **ambient lighting** (sunny day vs. overcast),
- **surface material and ink type** (matte paper vs. glossy foil),
- **sensor height** above the surface,
- **fill-light LED colour** on the sensor.

The figure on the right shows how, in addition to the reflected light we want to measure (green arrows), the sensor is disturbed by ambient light (red arrows) from the sun or lamps.

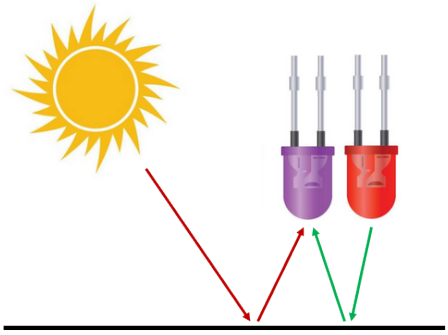


Figure 3. Effect of ambient light on sensor readings

Note

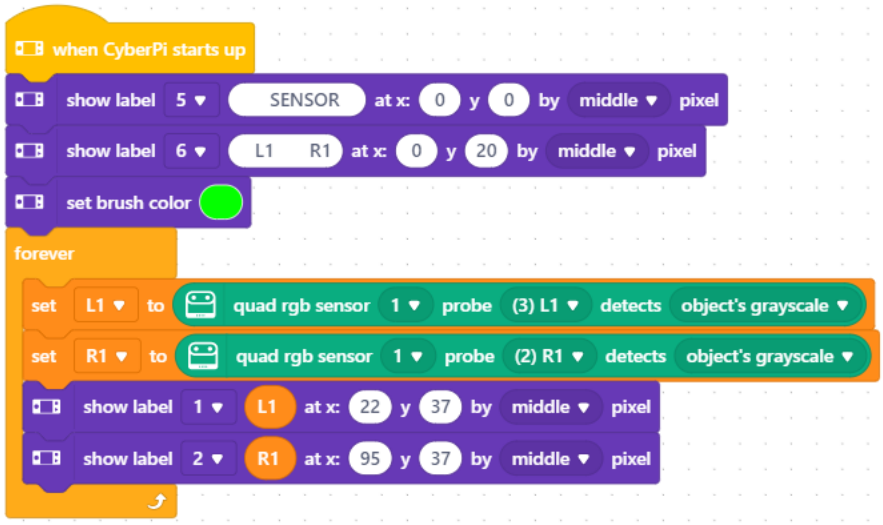
Without calibration a sensor that works perfectly in one room may fail in another – even with identical code.

The robot worked well in class but failed to follow the line at home using the same program. Name 2 possible causes.

6 Comparing Readings Before and After Calibration

If you are short on time, you can skip this section. However, if you want to really understand the process, it is worth recording sensor values before calibration. Upload the following program, which displays the reflected light intensity of the two central sensors L1 and R1.

Blockly – sensor experiment



Python – sensor experiment

```
import cyberpi, mbuild

L1 = 0
R1 = 0

cyberpi.display.show_label("      SENSOR   ",16,0,0,1)
cyberpi.display.show_label("   L1       R1",16,0,20,2)
cyberpi.display.set_brush(255, 255, 0)
while True:
  L1 = mbuild.quad_rgb_sensor.get_gray("L1",1)
  R1 = mbuild.quad_rgb_sensor.get_gray("R1",1)
  cyberpi.display.show_label(L1,16,22,37,3)
  cyberpi.display.show_label(R1,16,95,37,4)
```

6.1. Experiment

Place the robot over white paper and over the black line and record the values in the table below.

	Before calibration		After calibration	
	L1	R1	L1	R1
Value on white surface				
Value on black line				
Difference (white – black)				
Does the robot follow the line?				

Return to this table **after calibration** (or after measuring under different lighting) and record the new values, then compare.

7 Calibration Procedure – Method A (button)

This is the fastest method – no computer needed. You will need a flat surface with normal room lighting (the same you will use during the session) and white paper with a black line, e.g. the mBot2 track poster.

Steps:

1. Place the robot on the **white area** of the paper.
2. Check that the sensor is **12–13 mm** above the surface.
3. **Double-press** the button on the sensor.
4. The LEDs start blinking in different colours – calibration is in progress.
5. Slowly slide the robot so the sensor **passes over the black line and back** to white.
6. When the LEDs stop blinking, calibration is complete.

The sensor is now calibrated to distinguish the line from the background.

Warning

Do not calibrate under direct sunlight or very dim lighting – the results will be inaccurate.

8 Calibration Procedure – Method B (program)

Calibration can also be triggered from your program. This is useful if you want to repeat calibration automatically or verify it via mBlock.

Python – start calibration

```
import cyberpi, mbuild

mbuild.quad_rgb_sensor.adjust(1)
cyberpi.display.show_label("Done", 12, "center", index= 0)
```

9 Verifying Calibration – Experiment

After calibration we can verify that the sensor correctly distinguishes black and white. Upload the program below and slowly move the robot across the line. This time the display will show binary values 0 (white) and 1 (black) for each of the four sensors.

Python – verify calibration

```
import cyberpi, mbuild

cyberpi.display.show_label("  RGB SENSOR\n\n L2  L1  R1  R2\n", 16, 0, 0, index =
    0)
cyberpi.display.set_brush(255, 255, 0)

while True:
    L2 = mbuild.quad_rgb_sensor.is_line("L2",1)
    L1 = mbuild.quad_rgb_sensor.is_line("L1",1)
    R2 = mbuild.quad_rgb_sensor.is_line("R2",1)
    R1 = mbuild.quad_rgb_sensor.is_line("R1",1)

    vL1 = 1 if L1 else 0
    vL2 = 1 if L2 else 0
    vR1 = 1 if R1 else 0
    vR2 = 1 if R2 else 0

    cpi.display.show_label(vL2, 16, 10, 54, index = 1)
    cpi.display.show_label(vL1, 16, 45, 54, index = 2)
    cpi.display.show_label(vR1, 16, 75, 54, index = 3)
    cpi.display.show_label(vR2, 16, 110,54, index = 4)
```

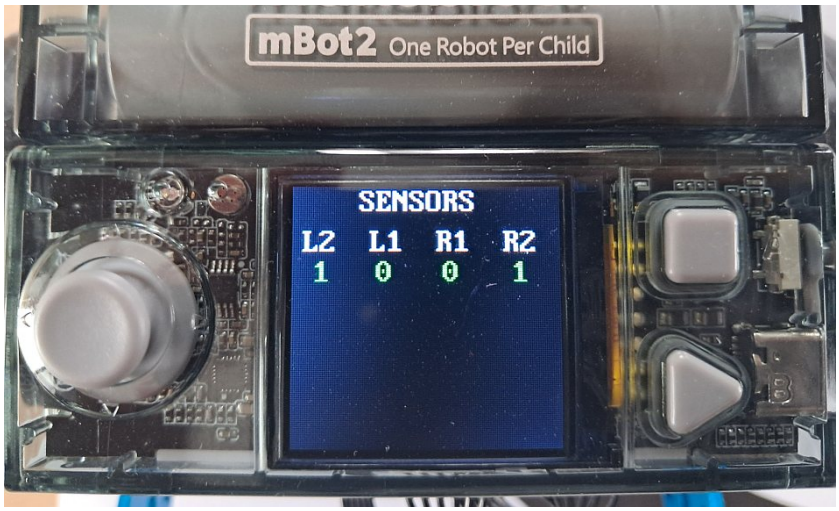


Figure 4. Display view when the robot is on the line

Why is it important to calibrate the sensor in the same location and under the same lighting where the robot will be used?

9.1. Verification Table

Slide the robot across the track and record what the display shows:

Robot position	L2	L1	R1	R2	Correct?
Entire robot on white	0	0	0	0	✔
Line under L1 and R1 (centre)	0	1	1	0	
Line under L1 only					
Line under R1 only					
Entire robot on black					
Other:					
Other:					

If the sensor shows value 0 on a black line instead of 1, what does this mean and what should you do?

10 Summary – Rules for Good Calibration

The larger the difference between the white-surface value and the black-line value, the better the calibration. Why?

Fill in the missing words:

1. Always calibrate under _____ lighting, matching the conditions during use.
2. The sensor height above the surface should be _____ to _____ mm.
3. If the room lighting changes significantly, the calibration must be _____.
4. A successful calibration is confirmed when the white surface reads _____ and the black line reads _____.

Note

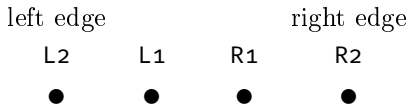
It is good practice to repeat calibration whenever you move the robot to a different room or when the ambient light changes significantly.

Name: _____ **Class:** _____ **Date:** _____

11 Recap – Line Sensor

In the previous session we got to know the Quad RGB sensor and learned how to calibrate it. Today we will use it for our first real line-following algorithm.

Let us recall the arrangement of the sensing elements (top view):



Note

Today we will work only with the two **central** sensors – **L1** and **R1**. The outer sensors L2 and R2 will be used in the next session.

Each sensor returns one of two values:

Value	Meaning
`True` (1)	sensor detects **black line**
`False` (0)	sensor detects **white surface**

12 Bang-Bang Algorithm

Bang-Bang is the simplest form of control – the robot does not react smoothly but switches between fixed actions depending on what the sensor currently sees. No maths, no calculations – just a series of `if/elif` conditions.

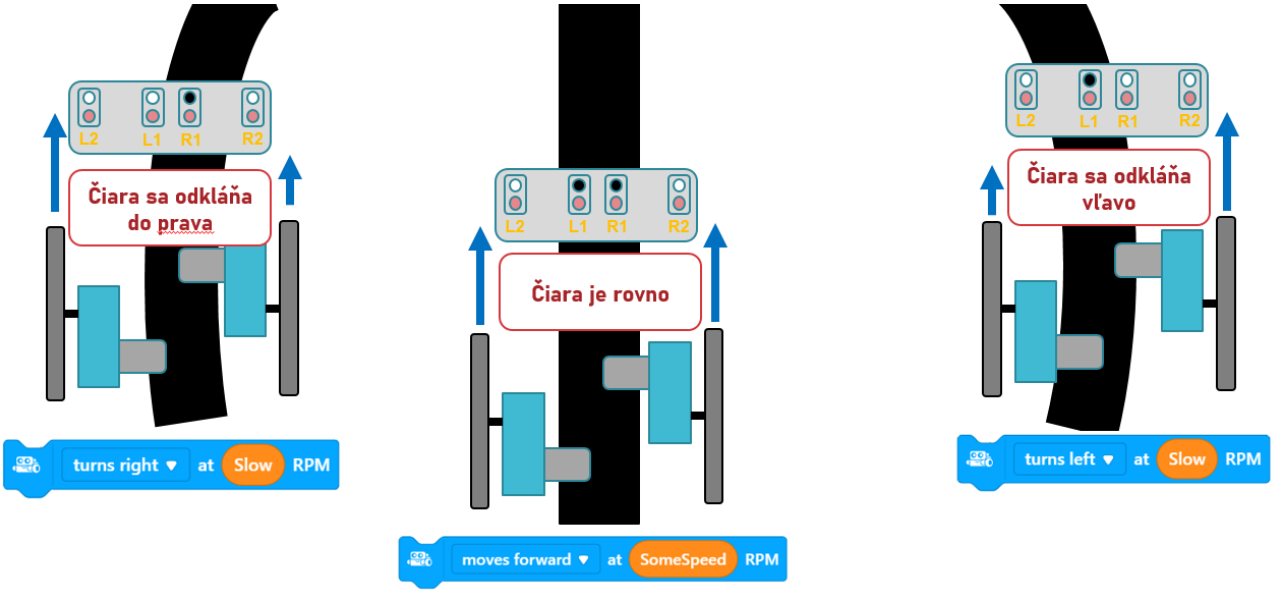


Figure 5. Simplest algorithm for line following

12.1. State Table

The combination of L1 and R1 values determines what the robot does:

L1	R1	State	Situation	Robot action
1	1	3	line in centre	go straight
1	0	2	line more to the left	turn left
0	1	1	line more to the right	turn right
0	0	0	line lost	stop

12.2. Pseudocode

Based on the state table, let us first write the algorithm in plain language (think it through carefully):

Pseudocode

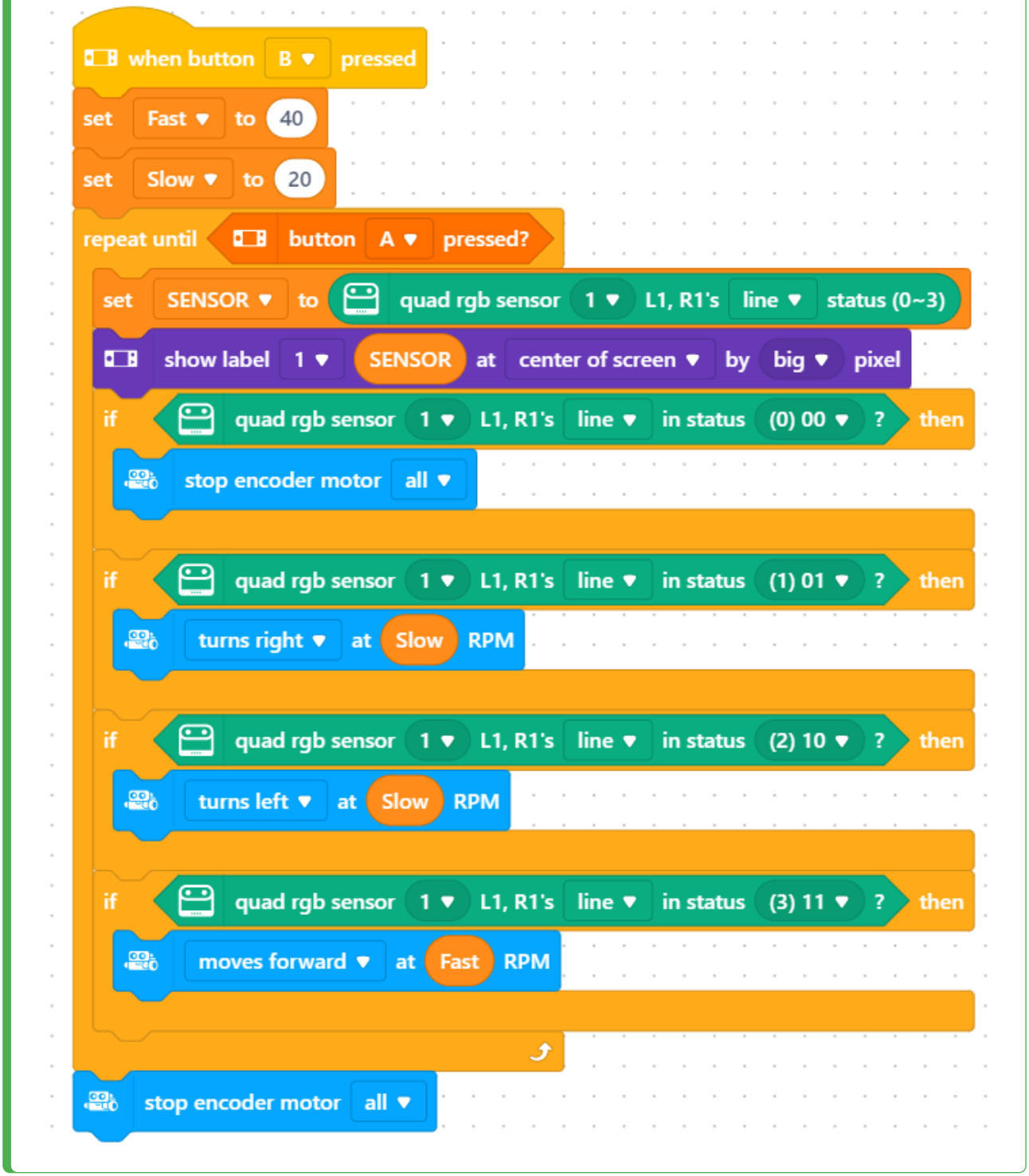
```
repeat forever:
  L1 = value of left central sensor
  R1 = value of right central sensor

  if L1=1 and R1=1:  go straight
  else if L1=1 and R1=0:  turn left
  else if L1=0 and R1=1:  turn right
  else:  stop
```

Draw a flowchart of this algorithm. Remember to include: START, the REPEAT FOREVER loop, and all four branches.

13 Program

13.1. Blockly

Blockly – Bang-Bang, 2 sensors



13.2. Python

Python – Bang-Bang, 2 sensors

```python
import event, time, cyberpi, mbuild, mbot2

SlowSpeed = 0
FastSpeed = 0
Sensor = 0

@event.is_press('b')
def is_btn_press():
    global SlowSpeed, FastSpeed, Sensor
    FastSpeed = 40
    SlowSpeed = 20

    while not cyberpi.controller.is_press('a'):
      Sensor = mbuild.quad_rgb_sensor.get_line_sta("middle", 1)
      cyberpi.display.show_label(Sensor, 48, "center", index= 0)

      if mbuild.quad_rgb_sensor.get_line_sta("middle", 1) == 3:
        mbot2.forward(FastSpeed)

      if mbuild.quad_rgb_sensor.get_line_sta("middle", 1) == 2:
        mbot2.turn_left(SlowSpeed)

      if mbuild.quad_rgb_sensor.get_line_sta("middle", 1) == 1:
        mbot2.turn_right(SlowSpeed)

      if mbuild.quad_rgb_sensor.get_line_sta("middle", 1) == 0:
        mbot2.EM_stop("ALL")
```

Note on speeds

The program uses two different speeds: one for going straight (FastSpeed) and one for turning towards the line (SlowSpeed).

14 Testing and Tuning

14.1. Before You Start – Predict

Will the robot follow the line better with a higher or lower turning speed (SlowSpeed)? Why?

What happens if you set both speeds to 100?

14.2. Measurements Table

Try different speed combinations and record your results. Rate each combination on a scale of 1–5 (as in school):

Forward speed FastSpeed	Turning speed SlowSpeed	Result (1–5)	Notes
40	20		
50	30		
70	40		
10	50		
own: _____	own: _____		

15 Reflection

What is the main drawback of the Bang-Bang algorithm? Describe a situation where the robot would fail.

The robot "zigzags" – it oscillates from side to side. Suggest a way to improve this without changing the algorithm (i.e. only by adjusting the numbers).

Imagine you have all 4 sensors available (L2, L1, R1, R2). What new situations could you distinguish compared to today's 2-sensor version? Name at least 2.

Bonus challenge

Modify the program so that when the line is lost (state `0,0`) the robot spins in place and searches for the line instead of stopping. Write the pseudocode for this solution.

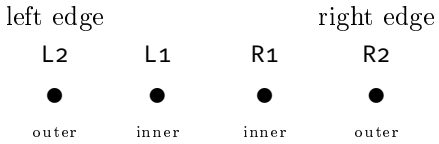
Name: _____ **Class:** _____ **Date:** _____

16 Recap and Motivation

In the previous session we programmed the robot using the Bang-Bang algorithm with two central sensors L1 and R1. The robot worked, but had one significant weakness.

What happened when the robot reached a sharp curve or when the line "escaped" both sensors at the same time? How did the robot behave?

Today we will also use the **outer sensors L2 and R2**. The robot will have a wider "field of view" – it will detect the line before losing it completely, and can react to larger deviations.



17 Real Sensor States

With four sensors there are theoretically $2^4 = 16$ different combinations. However, not all of them occur during normal driving along a continuous line.

Think it through and fill in the missing robot actions in the last column:

Status	L2	L1	R1	R2	Situation	Robot action
0	0	0	0	0	no line / end of track	stop
1	0	0	0	1	line far right	_____
3	0	0	1	1	line slightly right	_____
6	1	1	1	1	line in centre	go straight
12	1	1	0	0	line slightly left	_____
8	1	0	0	0	line far left	_____
15	1	1	1	1	all sensors on black	_____

Why can the state `L2=1, L1=0, R1=0, R2=1` (outer sensors see the line, inner sensors do not) not occur during normal driving along a single continuous line?

Think of 2 more states that are technically possible but would only occur in exceptional circumstances during normal line following. Explain when:

State _____: _____

State _____: _____

18 Pseudocode and Flowchart

18.1. Pseudocode

Pseudocode

```
repeat forever:
  L2, L1, R1, R2 = values of all four sensors

  if L2=1 and L1=1 and R1=1 and R2=1:  go straight
  else if L2=1 and L1=1 and R1=1 and R2=0:  gentle left turn
  else if L2=1 and L1=0 and R1=0 and R2=0:  sharp left turn
  else if L2=0 and L1=1 and R1=1 and R2=0:  gentle right turn
  else if L2=0 and L1=0 and R1=0 and R2=1:  sharp right turn
  else:  stop
```

Draw a flowchart of this algorithm. Use hexagons for conditions and rectangles for actions.

19 Program

Python – Bang-Bang, 4 sensors

```python
import event, time, cyberpi, mbuild, mbot2

while True:
    cyberpi.display.show_label(mbuild.quad_rgb_sensor.get_line_sta("all", 1), 24,
        "center", index= 0)
    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 6:
        mbot2.forward(50)

    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 8:
        mbot2.turn_left(50)

    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 12:
        mbot2.turn_left(30)

    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 3:
        mbot2.turn_right(30)

    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 1:
        mbot2.turn_right(50)

    if mbuild.quad_rgb_sensor.get_line_sta("all", 1) == 0:
        mbot2.EM_stop("ALL")
```

20 Comparison with the 2-Sensor Version

20.1. Before Testing – Predict

In what ways do you expect an improvement over the 2-sensor version? Where do you not expect a difference?

20.2. Comparison Table

Try both versions on the same track and record your observations:

	2 sensors (ws02)	4 sensors (ws03)
Behaviour on straight line		
Behaviour on gentle curve		
Behaviour on sharp curve		
Behaviour when line is lost		
Overall rating (1–5)		

Did reality match your predictions? What surprised you?

21 Reflection

The 4-sensor Bang-Bang has more states but still only a "step" response. What would change if, instead of two fixed speeds, we could set the speed continuously based on how far the line is from the centre?

Name: _____ **Class:** _____ **Date:** _____

22 What is Wrong with Bang-Bang?

In the previous sessions we programmed the robot using the Bang-Bang algorithm. If you watched the robot carefully, you may have noticed an unpleasant behaviour.

What is the name of the phenomenon where the robot alternately overshoots the line on both sides and "zigzags"? What causes it?

Note

Bang-Bang does not know how far the line is – it only knows *which direction*. Today we will fix this: we will teach the robot to react **more strongly** when the line is far away, and **more gently** when it is close to the centre.

23 Part A – Experiment: Digital Sensor

23.1. Setup

The robot is switched off. Place it on the track so that the line passes directly under sensors L1 and R1 (centre). You will slide it slowly sideways and record sensor values.

Upload the verification program from Worksheet 1 (p. XX).

23.2. Measurement 1 – Digital Values

Slide the robot slowly from the far left to the far right. Stop at each position and record the values of L1 and R1:

Robot position	L1	R1	error = L1 − R1
Line far left (under L2)			
Line under L1			
Line in centre (L1 and R1)			
Line under R1			
Line far right (under R2)			

What different values can `error = L1 - R1` take with a digital sensor? List all possible values.

How many different error values did you get? Is that enough for smooth control? Why?

If motor speed were calculated as `speed = 50 + Kp * error`, how many different speeds could the motor have (for any value of Kp)? What does this mean for smoothness of movement?

Aha moment

A P-controller with a **digital** sensor is in fact just Bang-Bang written differently – it has the same limitation. We need a sensor that provides **more information**.

24 Part B – Experiment: Analogue Sensor

The analogue sensor does not just measure "black/white" but **reflected light intensity** – a number from 0 to 100.

Value	Meaning
close to 0	minimum reflected light = **black line**
close to 100	maximum reflected light = **white**

Switch to analogue reading with this program:

Python – display analogue values

```python
import cyberpi, mbuild

cyberpi.display.show_label("L1      R1", 16, 0, 0, index=0)
cyberpi.display.set_brush(255, 255, 0)

while True:
    L1 = mbuild.quad_rgb_sensor.get_gray("L1", 1)
    R1 = mbuild.quad_rgb_sensor.get_gray("R1", 1)

    cyberpi.display.show_label(L1, 16, 22, 37, index=1)
    cyberpi.display.show_label(R1, 16, 95, 37, index=2)
```

24.1. Measurement 2 – Analogue Values

Slide the robot as in Measurement 1. This time record analogue values (0–100) for L1 and R1. The sensor can also directly measure the deviation from the line centre; record this value (Dev) in the table too:

Robot position	L1	R1	error = R1 − L1	Dev
Line far left (under L2)				
Line between L2 and L1				
Line under L1				
Line in centre				
Line under R1				
Line between R1 and R2				
Line far right (under R2)				

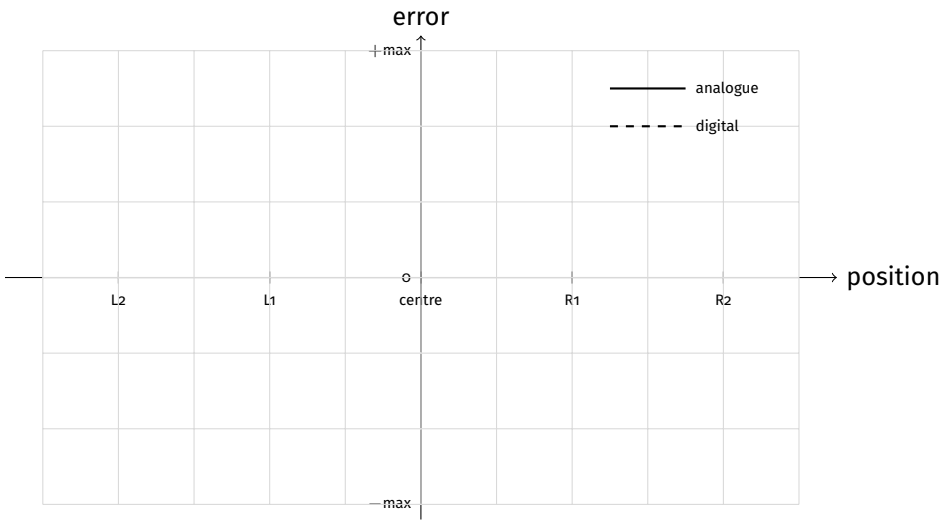
What is the minimum and maximum deviation (error) you measured?

Min. error: _____ **Max. error:** _____

Compare the number of different error values for the digital and analogue sensor. Which is better suited for smooth control and why?

24.2. Graph

Draw the **error as a function of robot position** for both sensors using your measured values.



What shape does the analogue sensor curve have? Does it remind you of anything from mathematics?

25 P-Controller

Now that we know the analogue sensor provides smooth error values, we can build a **proportional controller (P-controller)**.

25.1. Principle

The idea is simple: **the larger the deviation, the larger the correction.**

$$\texttt{correction} = K_p \times \texttt{error}$$

$$\texttt{left motor} = \texttt{base\_speed} - \texttt{correction}$$

$$\texttt{right motor} = \texttt{base\_speed} + \texttt{correction}$$

K_p (proportional gain) determines how strongly the robot reacts to deviation. It must be tuned experimentally.

What happens when `error = 0` (line exactly in the centre)?

correction = _____ **left motor =** _____ **right motor =** _____

The line is far to the left, `error = +100`, base speed = 50, K_p = 0.3. Calculate:

correction = _____ **left motor =** _____ **right motor =** _____

Is the robot turning in the correct direction? _____

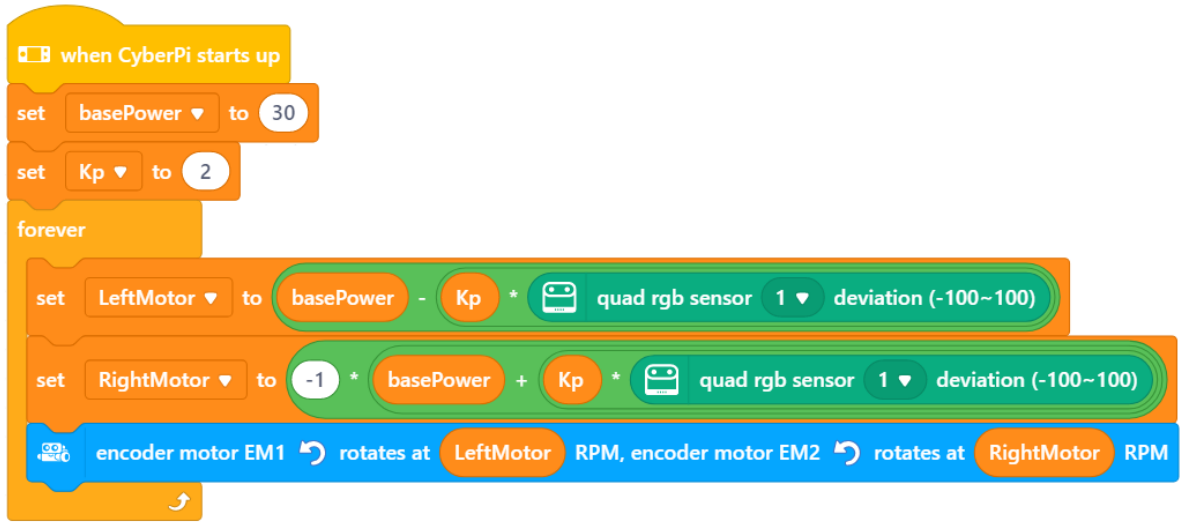
If we increase K_p from 0.3 to 1.5, what will happen to the robot's behaviour? Predict:

25.2. Program

Python – P-controller

```
import event, time, cyberpi, mbuild, mbot2


basePower = 30
Kp = 0.5
LeftMotor = 0
RightMotor = 0

while True:
    LeftMotor = (basePower - Kp * mbuild.quad_rgb_sensor.get_offset_track(1))
    RightMotor = -1 * ((basePower + Kp * mbuild.quad_rgb_sensor.get_offset_track
        (1)))
    mbot2.drive_speed(LeftMotor, RightMotor)
```

Blockly – P-controller



26 Tuning K_p

Try different values of K_p and record the robot's behaviour:

K_p	Behaviour on straight	Behaviour on curves	Rating (1–5)
0.1			
0.3			
0.5			
1.5			
2.0			
own: _____			

Find the value of K_p at which the robot performs best. Does the P-controller have any remaining weakness – does the robot always behave perfectly?

27 Summary and Comparison

Rate each algorithm on a scale of 1 (best) to 5 (worst) based on your own testing:

	Bang-Bang (2)	Bang-Bang (4)	P-controller
Straight line			
Gentle curve			
Sharp curve			
Smoothness			
Program complexity			

What is the most important advantage of the P-controller over Bang-Bang?

Next step

The P-controller only reacts to the **current** deviation, not to how quickly it is changing. If the robot overshoots the line and the deviation is growing fast, the P-controller cannot respond in time. This is solved by the **PD-controller** – we add a derivative term that reacts to the *change* in error. But that is a topic for the next session!