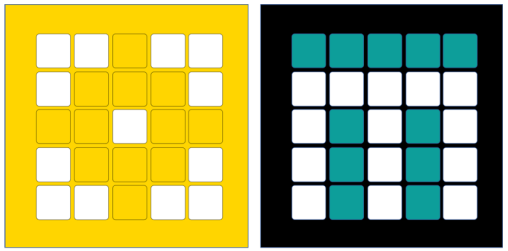
PRIME LESSONS

By the Makers of EV3Lessons



PID LINE FOLLOWER

BY SANJAY AND ARVIND SESHAN

This lesson uses SPIKE 3 software

LESSON OBJECTIVES

Learn the limitations of proportional control

Learn what PID means

Learn how to program PID and how to tune

WHEN DOES PROPORTIONAL CONTROL HAVE TROUBLE?

Note: the following few slides are animated. Use PowerPoint presentation mode to view them

What would a human do?

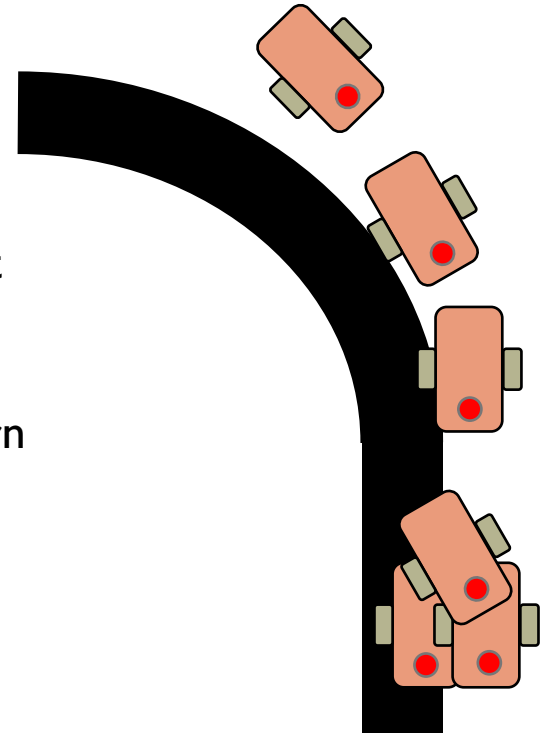
On line - go straight

On white - turn left

Moving across line - turn right

On white - turn left

Getting further from line - turn even more!



What would proportional control do?

On line - go straight

On white - turn left

Moving across line - go straight!

On white - turn left

Getting further from line - turn left the same amount!

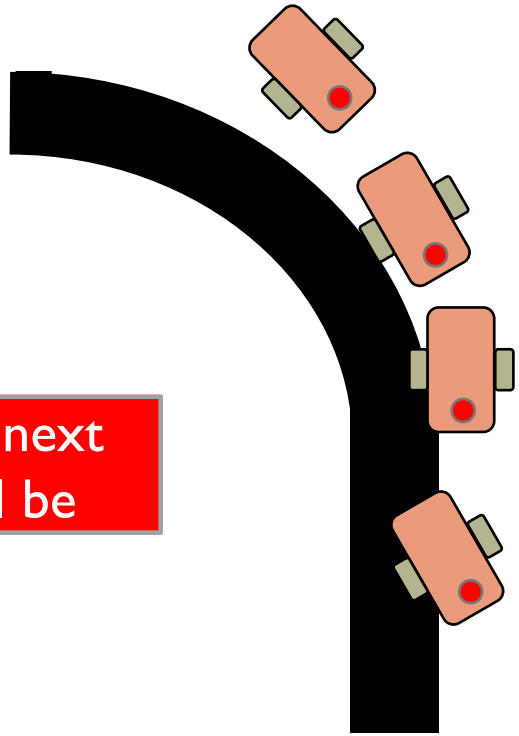
LIGHT READING = 50%

HOW CAN WE FIX PROPORTIONAL CONTROL?

What would a human do?

Turning left/on line ☐ turn right

Getting further from line turn even more!



What would proportional control do?

Turning left/on line go straight!

Getting further from line turn left the same amount!

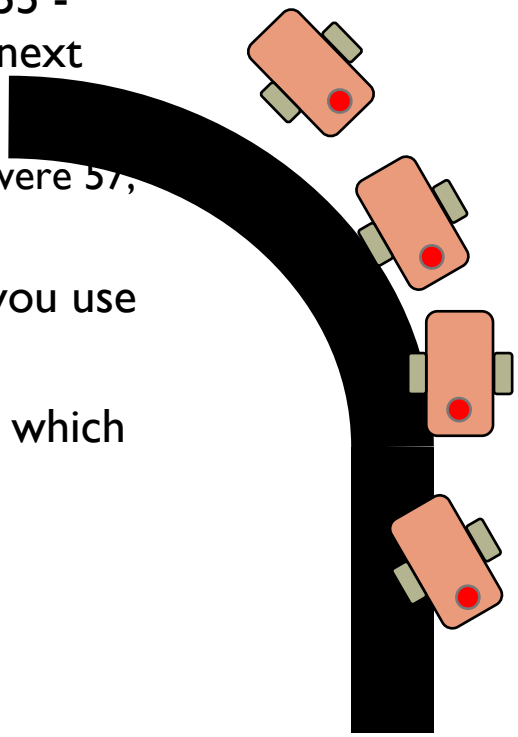
1. Predict what the next sensor reading will be

2. Has past steering fixes helped reduce error?

INTEGRALS AND DERIVATIVES

1. Predict what the next sensor reading will be?

- If readings are: 75, 65, 55 - what do you think the next reading will be?
  - What if the readings were 57, 56, 55…
- What information did you use to guess?
- Derivative - the rate at which a value is changing



2. Have past steering fixes helped reduce error?

- When the correction is working well, what does error readings look like?
  - +5, -6, +4 -3…. i.e. bouncing around 0
- When steering is not working, what does error look like?
  - +5, +5, +6, +5… i.e. always on one side of 0
- How can we detect this easily?
  - Hint: look at the sum of all past errors
- What is an ideal value for this sum? What does it mean if the sum is large?
- Integral - the "sum" of values

WHAT IS PID?

Proportional [Error] : How bad is the situation now?

Integral : Have my past fixes helped fix things?

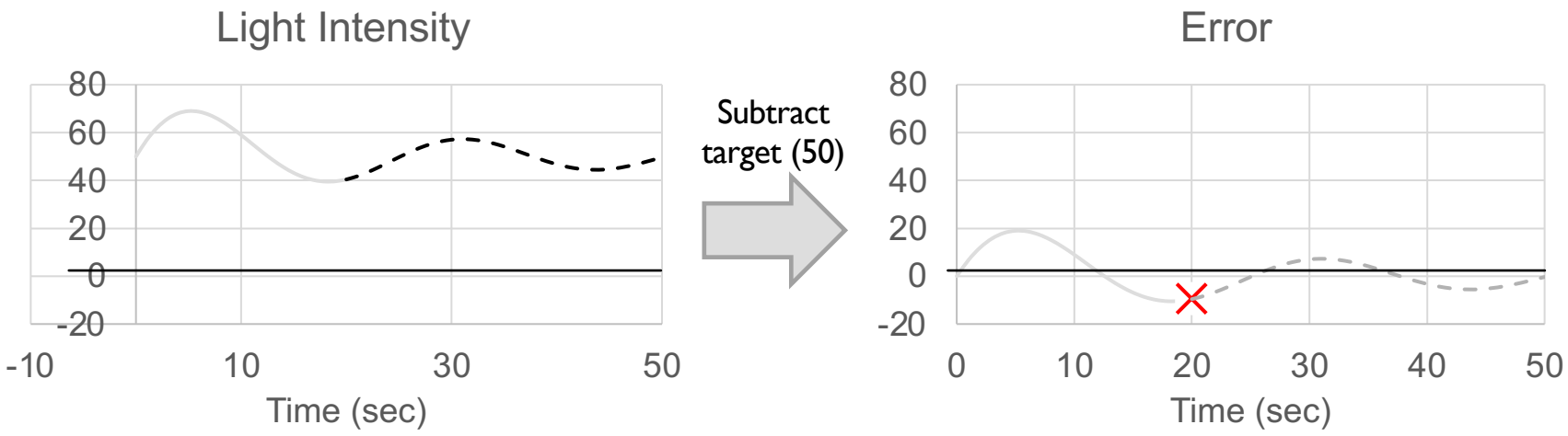
Derivative : How is the situation changing?

PID control : combine the error, integral and derivative values to decide how to steer the robot

ERROR

Solid line represents what you have seen, dotted line is the future

At time 20, you see light reading = 40 and error = -10 (red X)



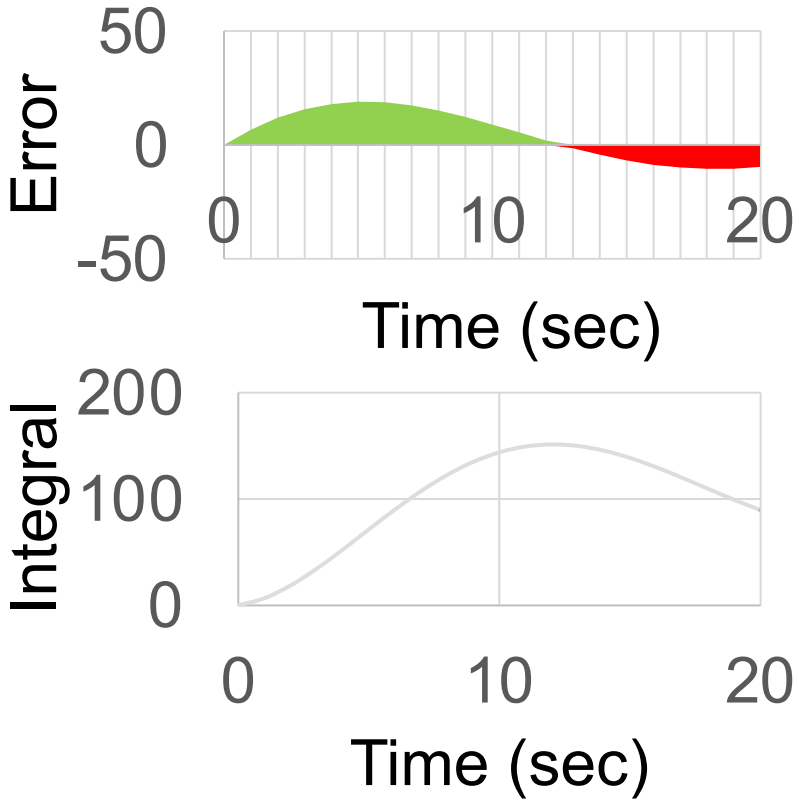
INTEGRAL

Looks at past history of line follower

Sum of past error

Like area under the curve in graph (integral)

- Green = positive area
- Red = negative area



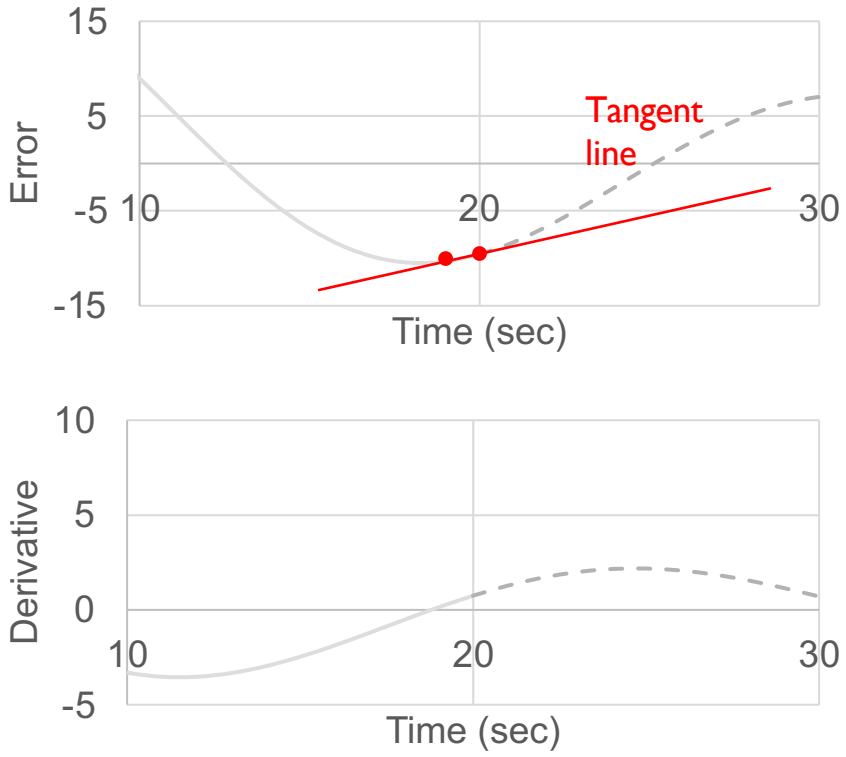
DERIVATIVE

How quickly is position changing?

- Predicts where the robot will be in the immediate future
- Same as how fast is error changing

Can be measured using tangent line to measurements: derivative

- Approximated using two nearby points on graph



PSEUDOCODE

1. Take a new light sensor reading
2. Compute the "error"
3. Scale error to determine contribution to steering update (proportional control)
4. Use error to update integral (sum of all past errors)
5. Scale integral to determine contribution to steering update (integral control)
6. Use error to update derivative (difference from last error)
7. Scale derivative to determine contribution to steering update (derivative control)
8. Combine P, I, and D feedback and steer robot

CODE - PROPORTIONAL

This is the same as the proportional control code, and follows a Black-White edge (right side of a black line)

Error = distance from line = target - reading

```
error = 50 - color_sensor.reflection(port.A)
P_fix = error * 0.5
```

Correction (P_fix) = Error scaled by proportional constant (K_p) = 0.5

CODE - INTEGRAL

This section calculates the integral. It adds the current error to a variable that has the sum of all the previous errors.

The scaling constant is usually small since Integral can be large

Integral = sum of all past errors = last integral + newest error

```
integral = integral + error # or integral+=error
I_fix = integral * 0.001
```

Correction (I_fix) = Integral scaled by proportional constant (K_i) = 0.001

CODE - DERIVATIVE

This section of code calculates the derivative. It subtracts the current error from the past error to find the change in error.

Derivative = rate of change of error = current error – last error

```
derivative = error - lastError
lastError = error
D_fix = derivative * 1
```

Correction (D_fix) = Derivative scaled by proportional constant (K_d) = 1.0

PUTTING IT ALL TOGETHER

Each of the components have already been scaled. At this point we can simply add them together.

Add the three fixes for P, I, and D together. This will compute the final correction

NOTE: Use the correction as steering after clamping it from -100 to 100 because SP3 does not appear to do it internally. i.e., it will accept values out of bounds (e.g. –250) and do unpredictable things.

```
correction = min(100, max(-100, int(P_fix + I_fix + D_fix)))
motor_pair.move(motor_pair.PAIR_1, correction, velocity = 300)
```

FULL CODE

```
from hub import port
import motor, motor_pair, color_sensor, runloop, sys

# Constants for Drive Base 1
motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)

# Follow the right side of black line (Black-White edge).
# To follow a White-Black edge, change the error condition to (reflection - 50).
async def pid_line_follow_forever():
    integral = 0
    lastError = 0
    while (True):
        # Compute the error.
        error = 50 - color_sensor.reflection(port.A)
        P_fix = error * 0.5
        integral = integral + error
        I_fix = integral * 0.001
        derivative = error - lastError
        lastError = error
        D_fix = derivative * 1
        # clamp the correction from -100 to 100 because SP3 doesn't seem to do it internally.
        correction = min(100, max(-100, int(P_fix + I_fix + D_fix)))
        # use the correction as the steering
        motor_pair.move(motor_pair.PAIR_1, correction, velocity = 300)

async def main():
    await pid_line_follow_forever()

runloop.run(main())
```

KEY STEP: TUNING THE PID CONSTANTS

The most common way to tune your PID constants is trial and error.

This can take time. Here are some tips:

Disable everything but the proportional part (set the other constants to zero). Adjust just the proportional constant until robot follows the line well.

Then, enable the integral and adjust until it provides good performance on a range of lines.

Finally, enable the derivative and adjust until you are satisfied with the line following.

When enabling each segment, here are some good numbers to start with for the constants:

P: 1.0 adjust by ±0.5 initially and ±0.1 for fine tuning

I: 0.05 adjust by ±0.01 initially and ±0.005 for fine tuning

D: 1.0 adjust by ±0.5 initially and ±0.1 for fine tuning

EVALUATING LINE FOLLOWERS

Proportional

Uses the "P" in PID

Makes proportional turns

Works well on both straight and curved lines

Good for intermediate to advanced teams - need to know math blocks

PID

It is better than proportional control on a very curved line, as the robot adapts to the curviness

However, for FIRST LEGO League, which mostly has straight lines, proportional control can be sufficient

CREDITS

This lesson was created by Sanjay Seshan and Arvind Seshan for Prime Lessons

Additional contributions by FLL Share & Learn community members.

More lessons are available at www.primelessons.org