definition and decomposition of problems in terms of functional requirements and constraints:
- define how the ultrasonic sensor works
- Use the Ultrasonic Sensor’s Wait for Change mode to detect proximity to an object
- Explain how to use the ultrasonic sensor
- Stop when an object is detected

### Design

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXPLORE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>GENERATE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DEVELOP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>EVALUATE &amp; IMPROVE</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Digital

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXPLORE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>GENERATE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DEVELOP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>EVALUATE &amp; IMPROVE</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Design</strong></th>
<th><strong>Digital</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>5-6 <strong>production</strong> of designed solutions by <strong>selecting</strong> and <strong>using</strong> appropriate technologies and techniques correctly and safely</td>
<td>5-6 <strong>definition</strong> of problems in terms of data and functional requirements</td>
</tr>
<tr>
<td>7-8 <strong>production</strong> of effective designed solutions for the intended purpose independently and safely</td>
<td>7-8 <strong>definition</strong> and <strong>decomposition</strong> of problems in terms of functional requirements and constraints</td>
</tr>
<tr>
<td>5-6 <strong>communication</strong> of design ideas to audiences using:</td>
<td>5-6 <strong>implementation</strong> of digital solutions, including a visual program</td>
</tr>
<tr>
<td>• graphical representation techniques</td>
<td>7-8 <strong>testing</strong>, <strong>modification</strong> and <strong>implementation</strong> of digital solutions</td>
</tr>
<tr>
<td>• technical terms</td>
<td></td>
</tr>
<tr>
<td>7-8 <strong>communication</strong> to different audiences using:</td>
<td></td>
</tr>
<tr>
<td>• appropriate technical terms</td>
<td></td>
</tr>
<tr>
<td>• a range of technologies and graphical representation techniques</td>
<td></td>
</tr>
<tr>
<td>5-6 <strong>evaluation</strong> of their ideas and designed solutions using their suggested criteria for success, including sustainability considerations</td>
<td>5-6 <strong>design</strong> of solutions by:</td>
</tr>
<tr>
<td>7-8 <strong>use</strong> of developed criteria for success (including sustainability considerations) to judge the suitability of:</td>
<td>• developing algorithms to address defined problems</td>
</tr>
<tr>
<td>• their ideas</td>
<td>• incorporating decision making, repetition (iteration) and user interface design</td>
</tr>
<tr>
<td>• designed solutions and processes</td>
<td>7-8 <strong>design</strong> of user experiences and algorithms incorporating branching and iterations</td>
</tr>
<tr>
<td>5-6 <strong>evaluation of information systems and their solutions in terms of meeting needs, innovation and sustainability</strong></td>
<td>7-8 <strong>evaluation of information systems and their solutions in terms of meeting needs, innovation and sustainability</strong></td>
</tr>
</tbody>
</table>

- functional requirements listed
- IPO table

Follow line

Write the Algorithm to Follow line

Which is better
How can you make it better

Which is better
How can you make it better
Target Path
Following a line works by ‘wiggling’ left and right along a target path.

The target or threshold light sensor value is calculated by adding the white and black sensor values together and dividing by 2.

Thresholds

- You need to be able to know what the Colour Sensor is seeing
- Place your robot so that the Colour Sensor is over White
- Right arrow 3 clicks to the right, select Port View
- Use the arrow buttons to see the Colour Sensor port
- Make sure its on Col-Reflect
- Record the reading
- Do the same for Black
- Calculate White + Black / 2
- This value is your White/Black threshold
Example Simple Line Follower

4. DEVELOP

4.1 Define Data and Functional Requirements

4.1.1 Functional Requirements
The robot will steer towards a threshold light value between light and dark

4.1.2 IPO Table

<table>
<thead>
<tr>
<th>Input (event/data)</th>
<th>Processing (coding)</th>
<th>Output (action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour Sensor value</td>
<td>If light</td>
<td>Steer towards dark (left)</td>
</tr>
<tr>
<td></td>
<td>If dark</td>
<td>Steer towards light (right)</td>
</tr>
</tbody>
</table>

4.2 Algorithm designs using pseudocode

**Threshold = 50**
Less than 50 is dark and greater is light colour

Assuming the colour sensor is right of the black line
If the value of the reflected light is greater than 40% (white or very light), our robot is outside the line, so steer to the left; change the turn ratio to -5 (left a bit), speed 15%.

Else, the reflected light is dark, so steer to the right; change the turn ratio to 15 (right a bit more), speed 5%. This slows down the robot and steers it to the right

IF Sensor = White THEN
  TurnLeft
ELSE TurnRight

From: [https://makecode.mindstorms.com/tutorials/line-following](https://makecode.mindstorms.com/tutorials/line-following)
Another Idea

Embedded systems perform a DO...UNTIL logic routine by using a pause or wait.

Forever
  DO
    Steer left -74 speed 20
    UNTIL colour sensor detects bright light

  DO
    Steer right 81 speed 20
    UNTIL colour sensor detects dark
Instructions

Code and test out the two algorithms above.

How much "wiggle" was there for each?

Which is better? Why?

Could you adjust the turn and speed settings to improve either? How?

A Better Line Follower

We need to reduce the wiggle
- Let's break the edge of the line up into three parts
- When the sensor is over White, turn left
- When the sensor is over Black, turn right
- When the sensor is halfway, which we can call Grey, go straight.

NOTE: you need to determine what these readings will be by using the port view on the brick.
4.1 Define Data and Functional Requirements

4.1.1 Functional Requirements

The robot will steer left away from light (>50), steer right away from dark (<40) and go forward when between 43 and 47

4.1.2 IPO Table

<table>
<thead>
<tr>
<th>Input (event/data)</th>
<th>Processing (coding)</th>
<th>Output (action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour Sensor value</td>
<td>If dark</td>
<td>Turn left</td>
</tr>
<tr>
<td></td>
<td>If light</td>
<td>Turn right</td>
</tr>
<tr>
<td></td>
<td>If between light and dark</td>
<td>Go straight</td>
</tr>
</tbody>
</table>

4.2 Algorithm designs using pseudocode

IF sensor reading > 50 THEN
    TurnLeft
ELSE IF sensor reading < 40 THEN
    TurnRight
ELSE IF sensor reading >= 43 AND <= 47 THEN
    GoStraight

Instructions

Code and test out the algorithms above.

Youtube: https://youtu.be/0CBMxbYmTlY
How much "wiggle" was there, compared to the simple versions?

Could you adjust the turn and speed settings to improve it? How?

Advanced - Proportional Line Following with one sensor

The control algorithm developed above is called a three-state controller because the program does one of three things based on the sensor reading: go straight, turn left, or turn right. The main problem with this method is that when the robot needs to turn, it always turns the same amount; whether the robot encounters a sharp or gentle turn, it uses the same fixed Steering value.

It would be better if the Steering value depended on the sharpness of the line, with gradual steering for straighter curves and sharp steering for sharper corners. This makes the program respond more quickly to changes in the direction of the line while still moving smoothly when the line is straight. This approach is called a proportional controller because the change made to the steering is proportional to, or directly related to, the robot’s distance from the edge of the line.

A proportional controller changes a control variable (in this case, the steering direction) based on a Target value and an Input value. In our case, the Input value is the reading from the Colour Sensor. The Target value is the Colour Sensor reading when the sensor is directly over the edge of the line. We determined this value by taking the average of the Colour Sensor readings with the Bot off the line and centred over the line, which in my case was 40.

The difference between the Target value and the Input value is called the Error value. You can think of the Error value as the difference between where we want the robot to be and where it actually is. We multiply the Error value by the Gain value to get the Turn ratio. The Gain value determines how quickly the robot reacts to changes in the Error value. A smaller gain makes the robot move slowly, which means that it might not react quickly enough for tight turns, but results in less side-to-side motion when the line is fairly straight. A larger Gain value means a quicker reaction but can cause jerkier motion. Selecting the Gain value is called tuning the controller and usually involves some trial and error.
These are the equations for the Error value and Turn ratio:

Error value = Target value – Sensor reading

TurnRatio = Error value × Gain value

\[ \text{turnRatio} = (\text{targetValue} – \text{colourSensor}) \times \text{gainValue} \]

Another Explanation: https://youtu.be/-AirqwC9DL4

MakeCode + EV3: Proportional line follower with 1 color sensor

From <https://www.youtube.com/watch?v=-AirqwC9DL4&feature=youtu.be>

4. DEVELOP

4.1 Define Data and Functional Requirements

4.1.1 Functional Requirements

A proportional controller changes a control variable (in this case, the steering direction) based on a **Target value** and an **Input value**. In our case, the Input
value is the reading from the Colour Sensor. The Target value is the Colour Sensor reading when the sensor is directly over the edge of the line. We determined this value by taking the average of the Colour Sensor readings with the Bot off the line and centred over the line, which in my case was 40.

The difference between the Target value and the Input value is called the Error value. You can think of the Error value as the difference between where we want the robot to be and where it actually is. We multiply the Error value by the Gain value to get the Turn ratio. The Gain value determines how quickly the robot reacts to changes in the Error value. A smaller gain makes the robot move slowly, which means that it might not react quickly enough for tight turns, but results in less side-to-side motion when the line is fairly straight. A larger Gain value means a quicker reaction but can cause jerkier motion. Selecting the Gain value is called tuning the controller and usually involves some trial and error.

4.1.2 IPO Table

<table>
<thead>
<tr>
<th>Input (event/data)</th>
<th>Processing (coding)</th>
<th>Output (action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color sensor value</td>
<td>Calculate turnRatio</td>
<td>Steer motors D + A TurnRatio speed 25</td>
</tr>
</tbody>
</table>

4.2 Algorithm designs using pseudocode

White reading = 68%
Black reading = 16%
targetValue = Threshold = (white + Black) /2 = 40
gainValue (40% of difference) = 0.7

on start
   targetValue = 40
   gainValue = 2

   forever
      error = targetValue - colourSensor
      turnRatio = error * gainValue
      Steer motors D + A turnRatio speed 20

Youtube: [https://youtu.be/aAcYedAq3PQ](https://youtu.be/aAcYedAq3PQ)
How did this work around a curved line?
Can you change the gain to improve it? How?

**Improving Curve Steering**

with addition of default constant speed

White reading = 68%
Black reading = 16%
Target value = Threshold = (white + Black) / 2 = 40
Default Constant Speed = 15
Gain (40% of difference) = 0.4

targetValue = Threshold = (white + Black) / 2 = 40
gainValue (40% of difference) = 0.7

Calculation for motor speed is the default constant speed that is maintained plus how fast we want to move to the line. Both the Default constant speed and the Gain can be experimented with to achieve the best outcome. You can also change these into constant variables so that changes are easier.

4. DEVELOP

4.1 Define Data and Functional Requirements

4.1.1 Functional Requirements

The robot will spin wheels left and right, back and forwards, depending on colour sensor readings and calculations for motor speeds:

\[
\text{motorR} = (\text{Colour sensor 1 reflected light} - \text{targetValue}) \times \text{gainValue} + \text{DefaultConstantSpeed} \\
\text{motorL} = (\text{targetValue} - \text{Colour sensor 1 reflected light}) \times \text{gainValue} + \text{DefaultConstantSpeed}
\]

4.1.2 IPO Table

<table>
<thead>
<tr>
<th>Input (event/data)</th>
<th>Processing (coding)</th>
<th>Output (action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colour Sensor value</td>
<td>Calculate L and R motors speeds and direction based on sensor readings</td>
<td>Run L and R motors independently based on speed calculations</td>
</tr>
</tbody>
</table>
Algorithm

on start
targetValue = 50
gainValue = 0.7
DefaultConstantSpeed = 15

Repeat Forever
    motorR = (Colour sensor 1 reflected light - targetValue) * gainValue + DefaultConstantSpeed
    motorL = (targetValue - Colour sensor 1 reflected light) * gainValue + DefaultConstantSpeed
    Run large motor D (left) at motorL speed
    Run large motor A (right) at motorR speed

How did this work around a curved line?

1 Colour Sensor Line Follower Page 11
How did this work around a curved line?

Can you change the gain or default speed to improve it? How?

Proportional Integral Derivative (PID) 1 Color Sensor Line Follower

For a thorough background on this, see: [https://www.throughtheclassroomdoor.com/pid-controlled-line-follower-robot/](https://www.throughtheclassroomdoor.com/pid-controlled-line-follower-robot/)

Formula for a PID Line Follower

target = light sensor value when half on white and half on black
Kp = fine tuning of proportional
Ki = fine tuning of integral
Kd = fine tuning of derivative
value = light sensor value
error = target-value
integral = integral + error
derivative = error - last error

\[
\text{turn ratio} = (\text{error} \times \text{Kp}) + (\text{integral} \times \text{Ki}) + (\text{derivative} \times \text{Kd})
\]

Target Path

Target Path Left sensor = 40
Target Path right sensor = 40

Proportional

Measures deviation from the target path (error) using the light sensor

\[
\text{turn ratio} = (\text{target} - \text{value}) \times \text{Kp}
\]

Remember, value is the light sensor value
Kp is the tuning variable and generally, the higher its value, the sharper the steering curves

Algorithm for Proportional

On Start
\[
\text{Target} = 40
\text{Kp} = 1
\]

Loop Forever

\[
\text{turnRatio} = (\text{Target} - \text{Reflected light value}) \times \text{Kp}
\text{Steer motors turnRatio speed 75}
\]
The sum of all error values

\[ \text{integral} = \text{integral} + \text{error} \]

Driving straight = 0 sum of error
Straying right = positive sum of error
Straying left = negative sum of errors

Ki is used to fine tune

**Algorithm for Proportional and Integral**

On Start

- Target = 40
- Kp = 1
- Ki = 1
- Error = 0
- Integral = 0

Loop Forever

- Error = Target - Reflected light value
- \( ? = \text{Error} \times Kp \)
- \( ? = \text{Integral} \times Ki \)

\( \text{turnRatio} = \)  
Steer motors turnRatio speed 75

**Derivative**

Uses past and current error values to anticipate the next error

Kd is used to fine tune

**Algorithm for Proportional, Integral and Derivative**

On Start

- Target = 40
- Kp = 1
- Ki = 1
- Error = 0
- Integral = 0
- lastError = 0
- Derivative = 0
- Kd = 1

Loop Forever

- Error = Target - Reflected light value
- Integral = Integral + Error
- Derivative = Error - lastError

\( \text{turnRatio} = (\text{Error} \times Kp) + (\text{Integral} \times Ki) + (\text{Derivative} \times Kd) \)
Steer motors turnRatio speed 75
lastError = Error

**Tuning**

**NOTE:** Light Sensor needs to be on the left side of the line. Otherwise, you need to also multiply turnRatio by -1

Step 1 - proportional
- set Kp = 1
- set Ki = 0, Kd = 0

This will focus on proportional and the lower this value, the smoother it should be.

Step 2 - Integral
- experiment with Ki until smooth

Step 3 - derivative
let lastError = 0
let turnRatio = 0
let Kd = 0
let derivative = 0
let Ki = 0
let integral = 0
let Kp = 0
let error = 0
let target = 0

target = 40
Kp = 1
Ki = 1
Kd = 1

error = target - sensors.color1.light(LightIntensityMode.Reflected)
integral = integral + error
derivative = error - lastError

turnRatio = error * Kp + (integral * Ki + derivative * Kd)
motors.largeBC.steer(turnRatio, 50)
lastError = error

forever(function () {...})