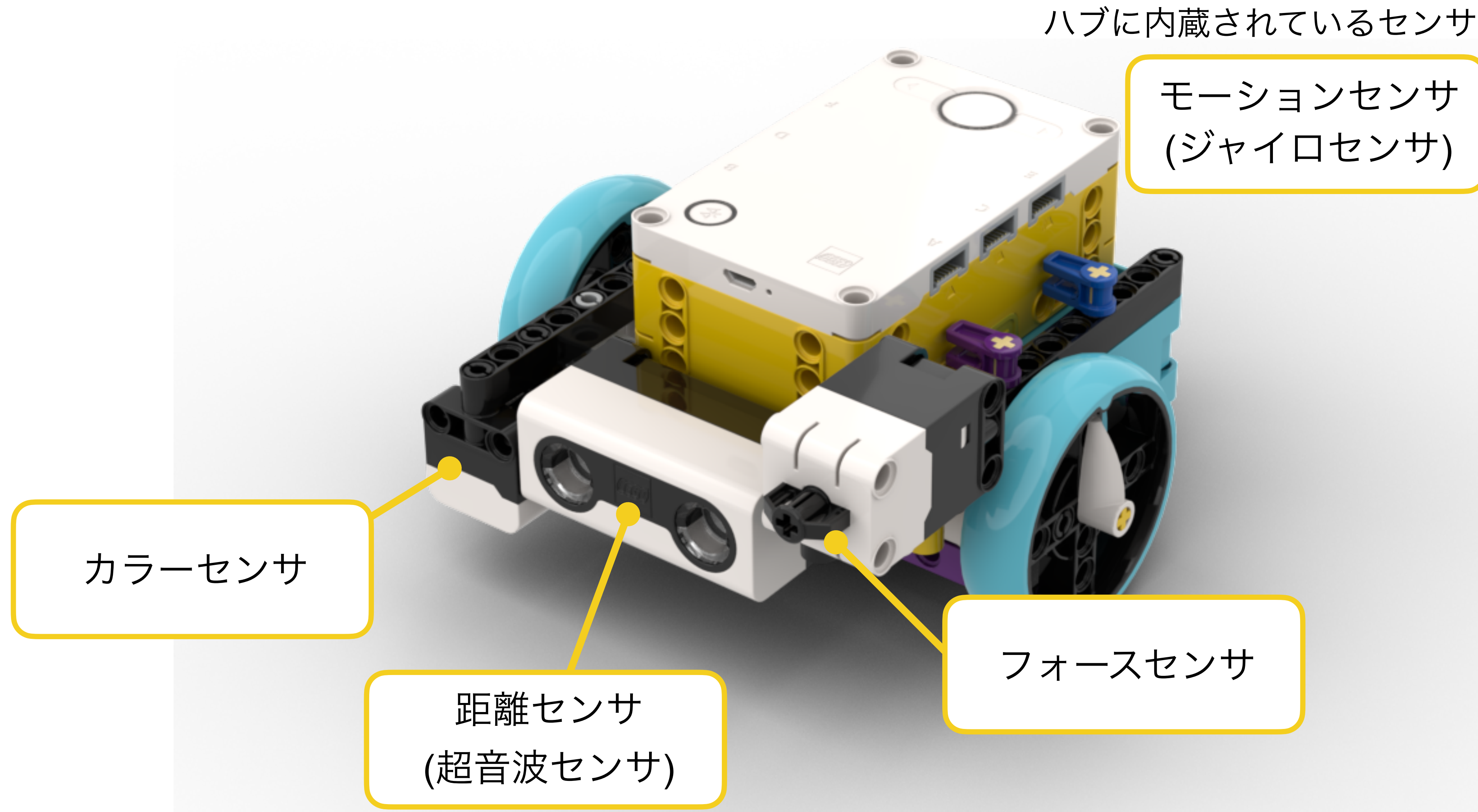


■ センサを使う

感じる、**判断する**、**動く**がそなわっている人工物

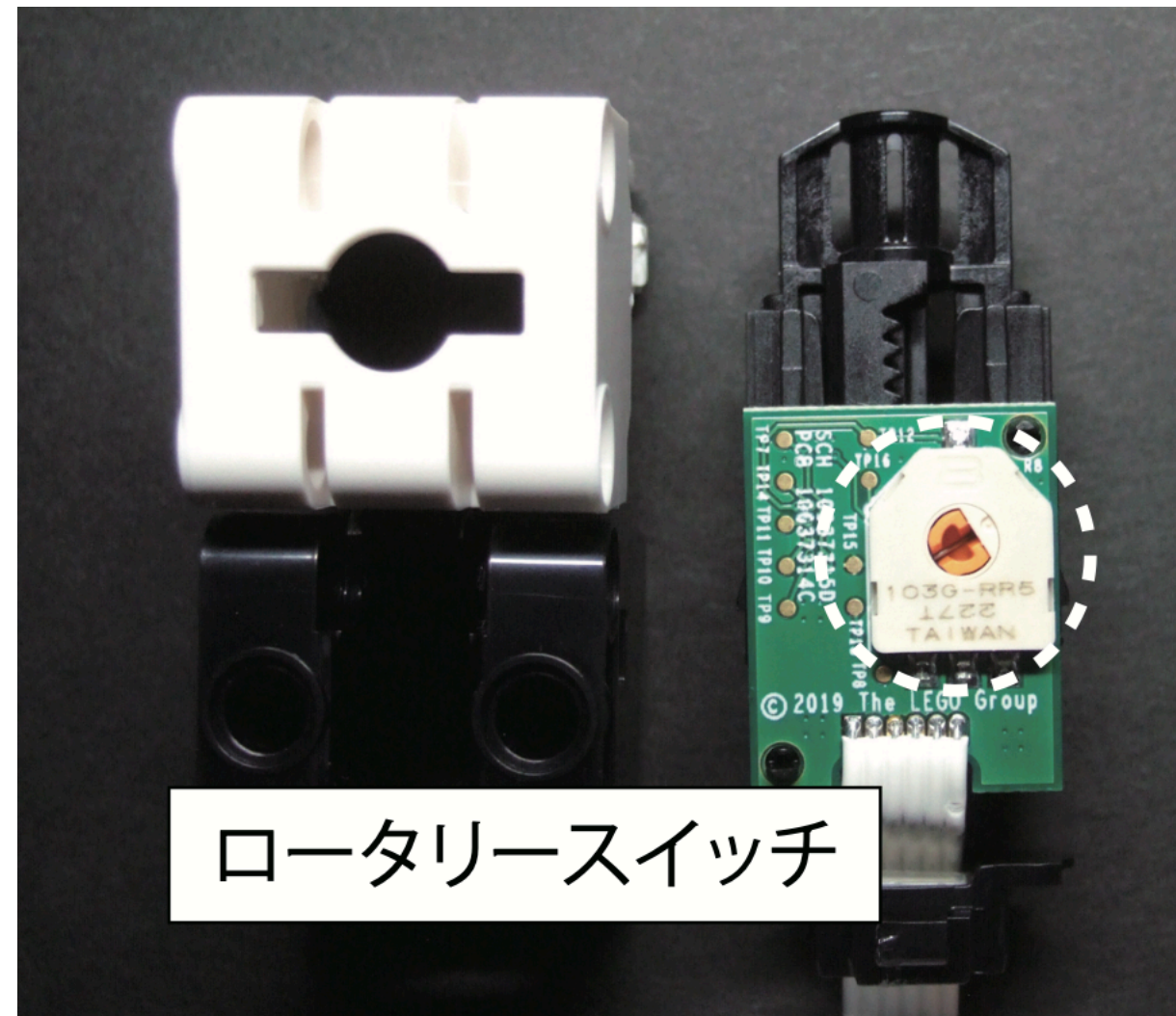
- 障害物回避(フォースセンサ)
- 障害物回避(距離センサ)
- カラーセンサによるライントレース

- ・ センサの名前を書きこもう！

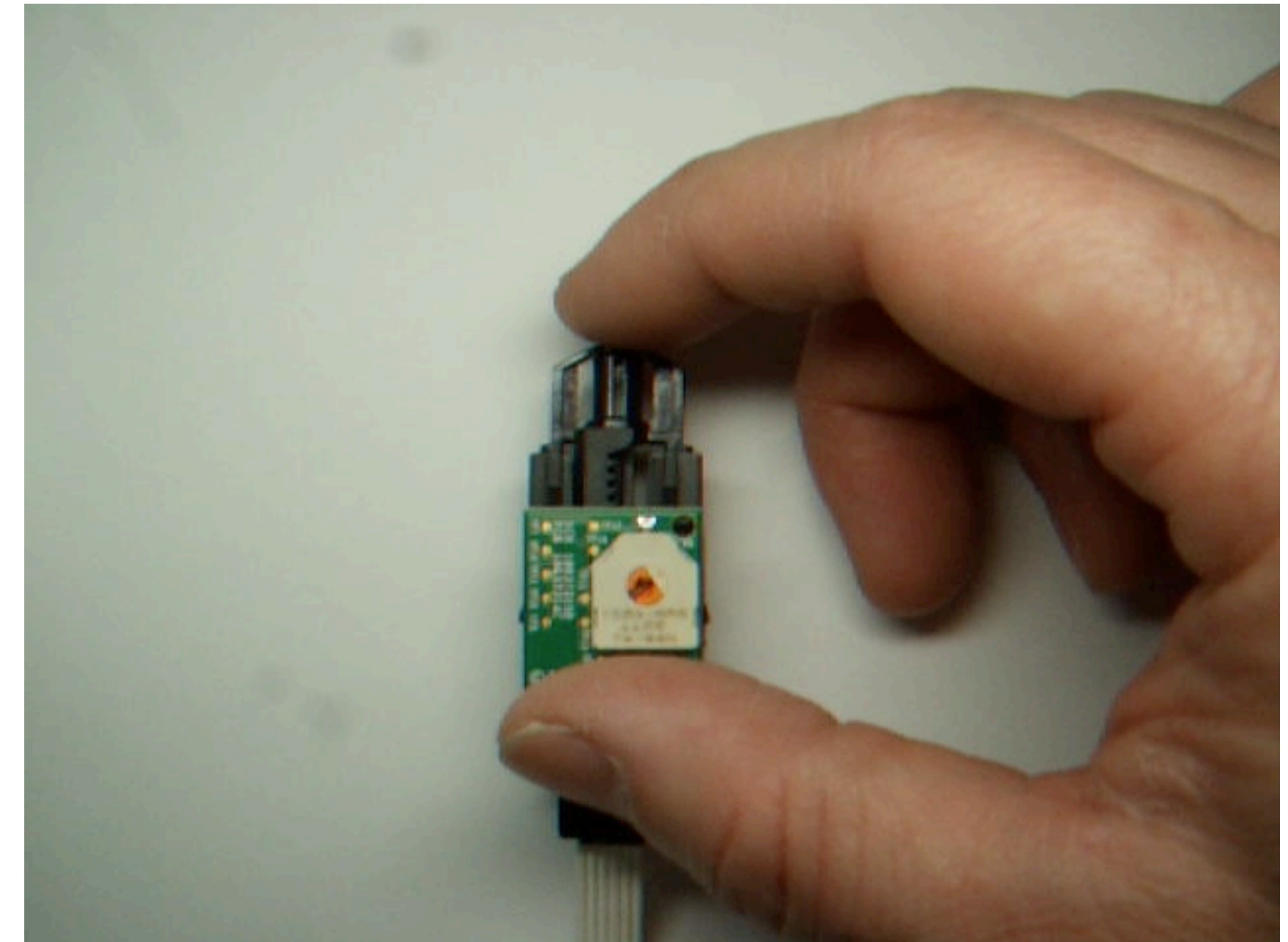
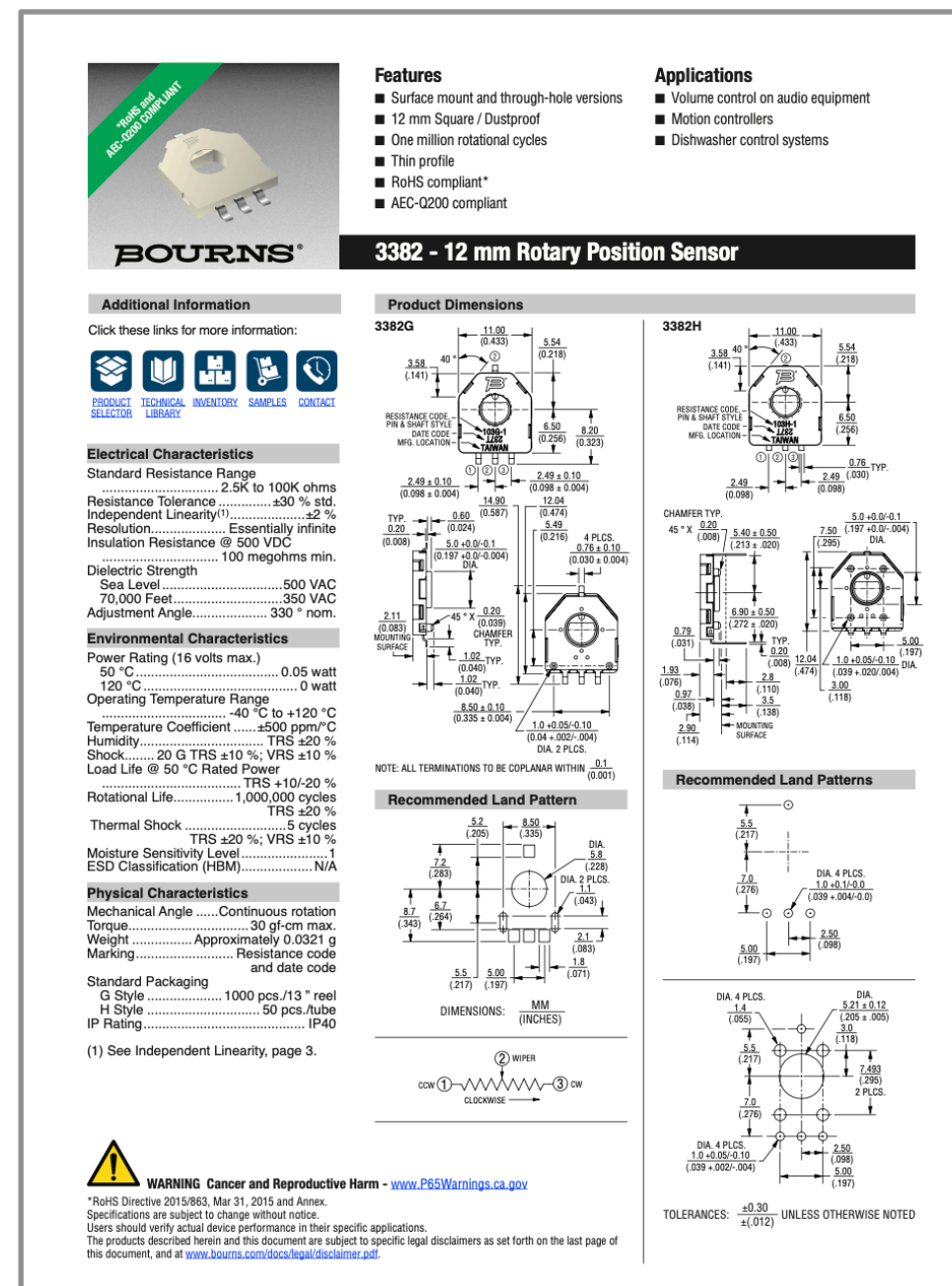


■障害物回避(フォースセンサ)

フォースセンサの原理



ロータリースイッチ



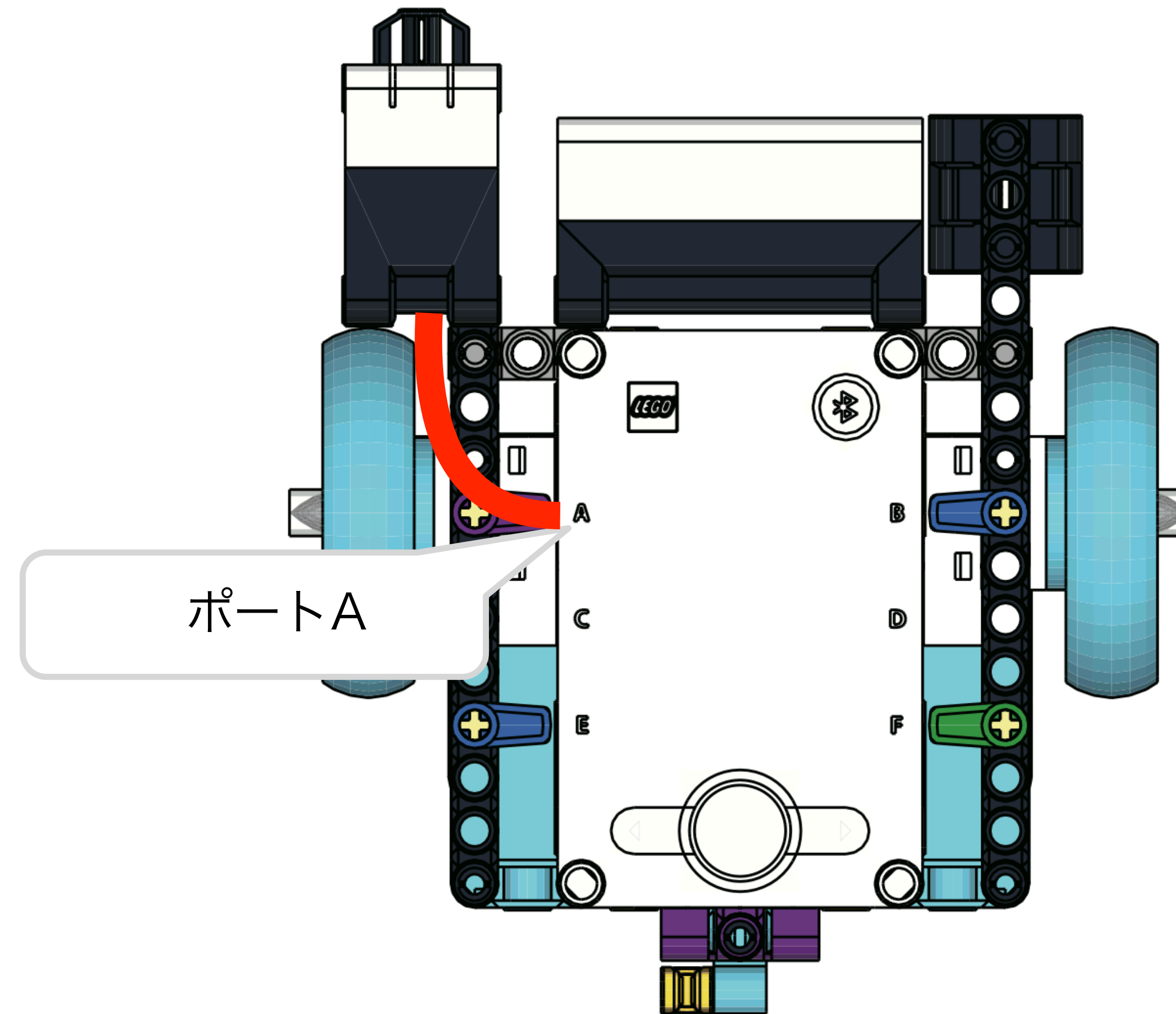
フォースセンサの測定原理

ロータリースイッチが内蔵されており、押す量を回転に変換する
これにより、「どのくらい押されているか」が正確に測定できる

フォースセンサの接続



- ・ ポートAにフォースセンサを接続



※SPIKEではポートにどのセンサを接続しているか自動判定しているため、必ずポートAである必要はない

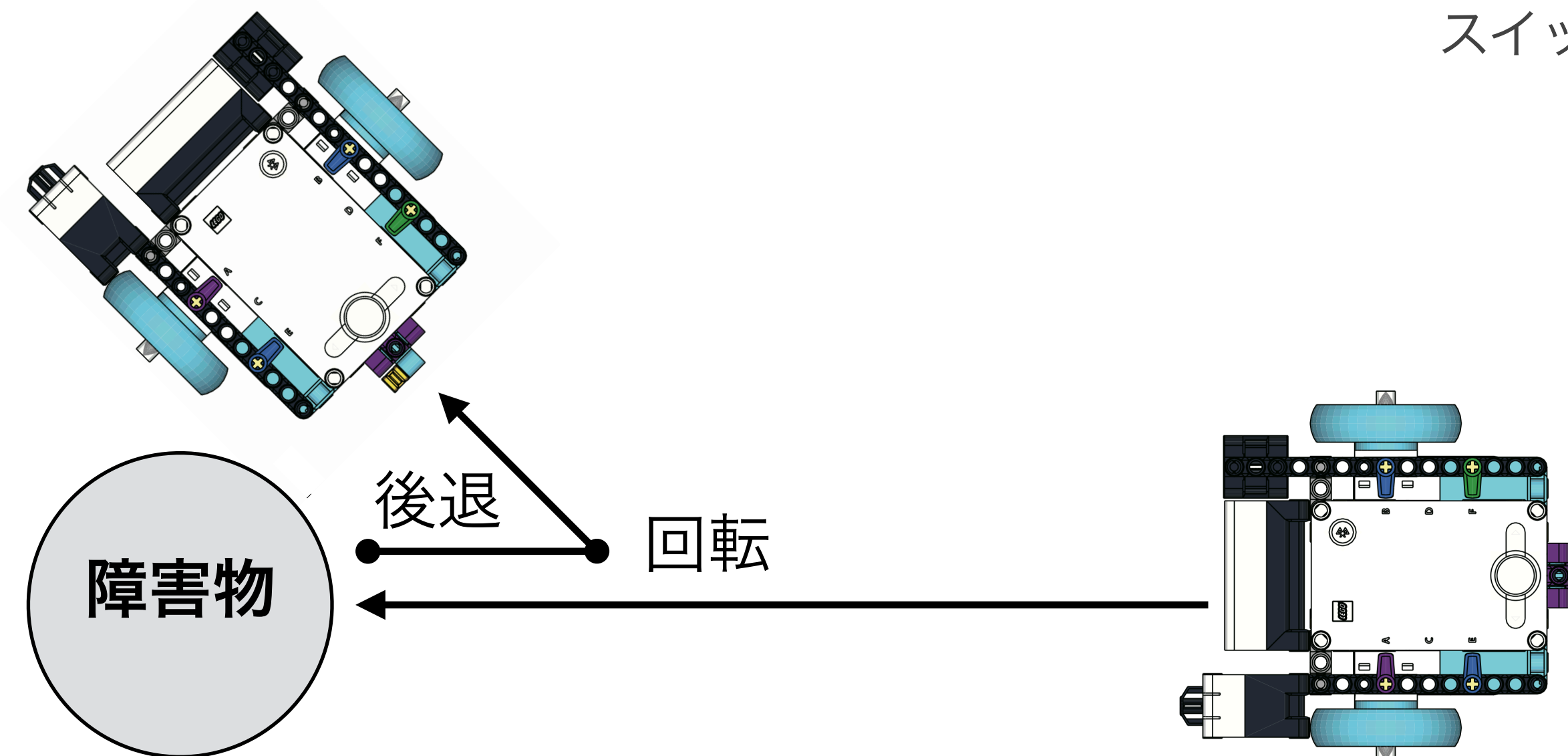
フォースセンサによる障害物回避



1. 常にロボットを前進 →無限ループ
2. フォースセンサによる障害物の衝突を判定
3. $\left\{ \begin{array}{l} \text{衝突あり：一定時間後退し右旋回. その後1.に戻る} \\ \text{衝突なし：1.に戻る} \end{array} \right.$



スイッチブロック

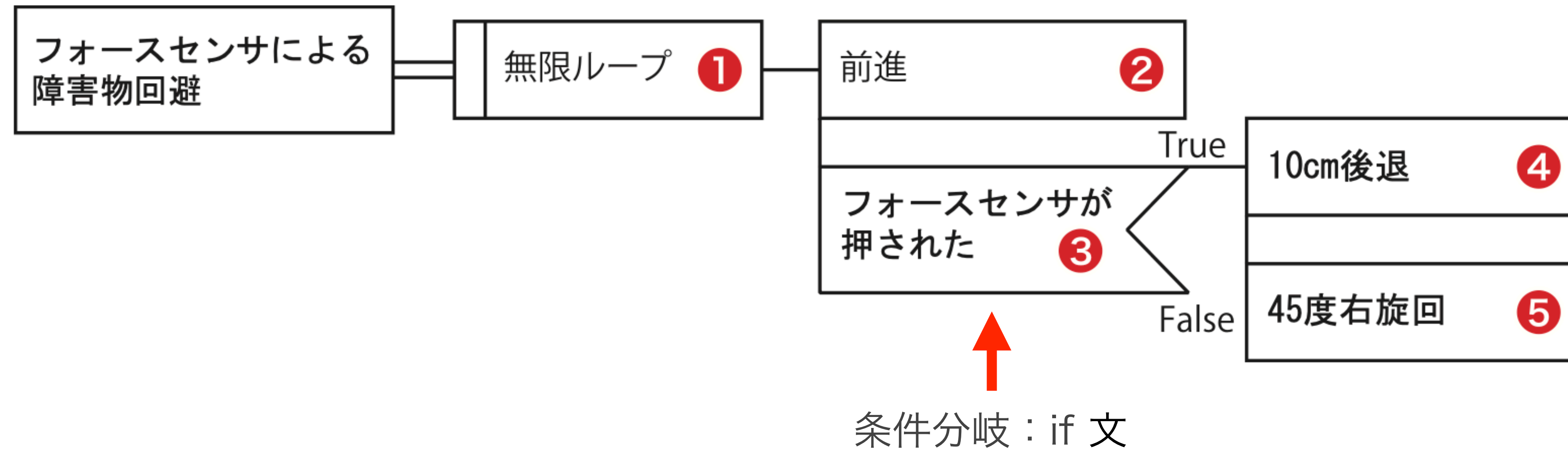


アルゴリズムを考えよう



簡単なアルゴリズム表現を使う

前進→フォースセンサが押されたら10cm後退→方向を変えて→前進



PADの構成部品：

処理：



選択：



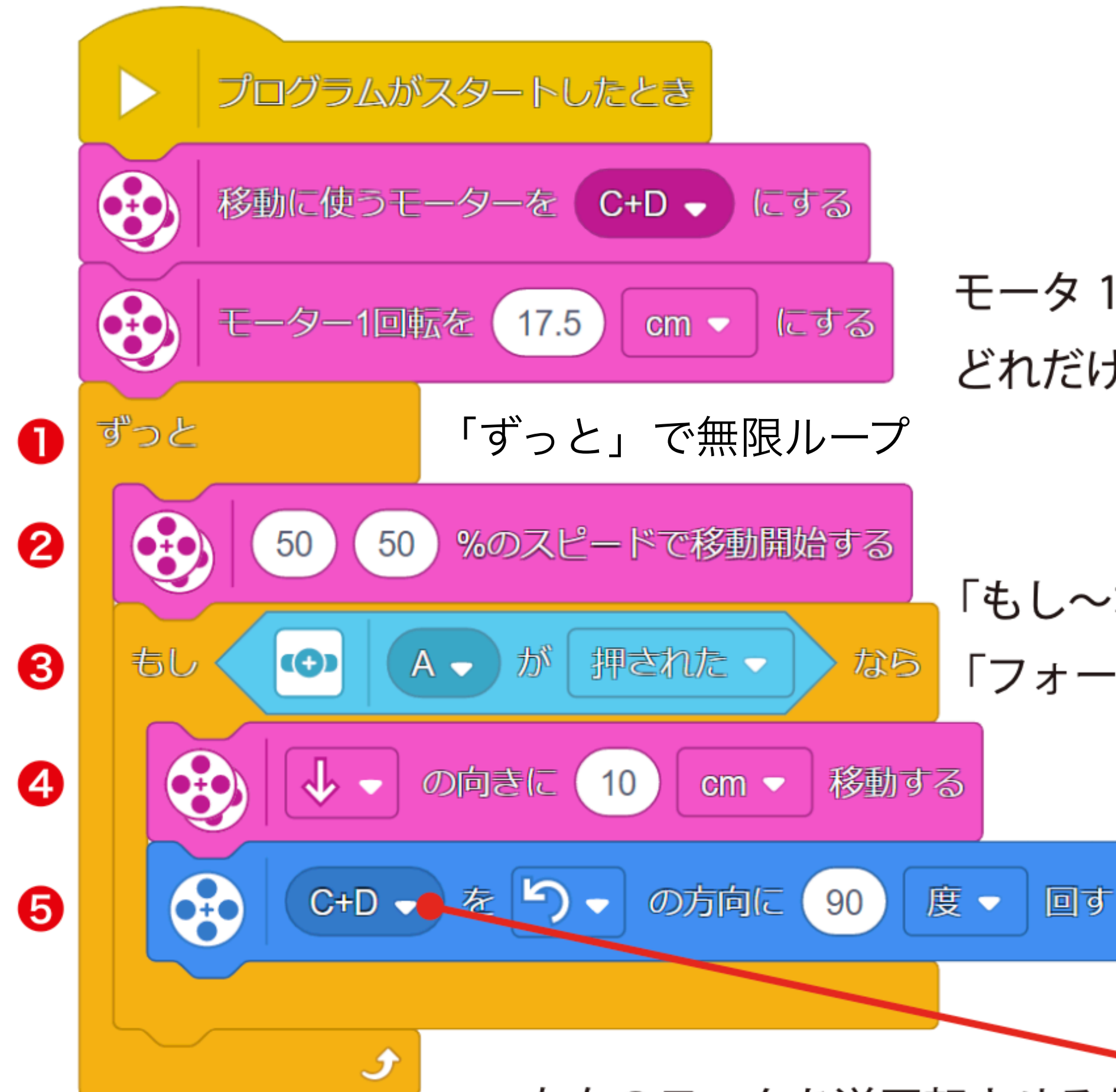
反復：



フォースセンサによる障害物回避



p.64 : force.llsp3

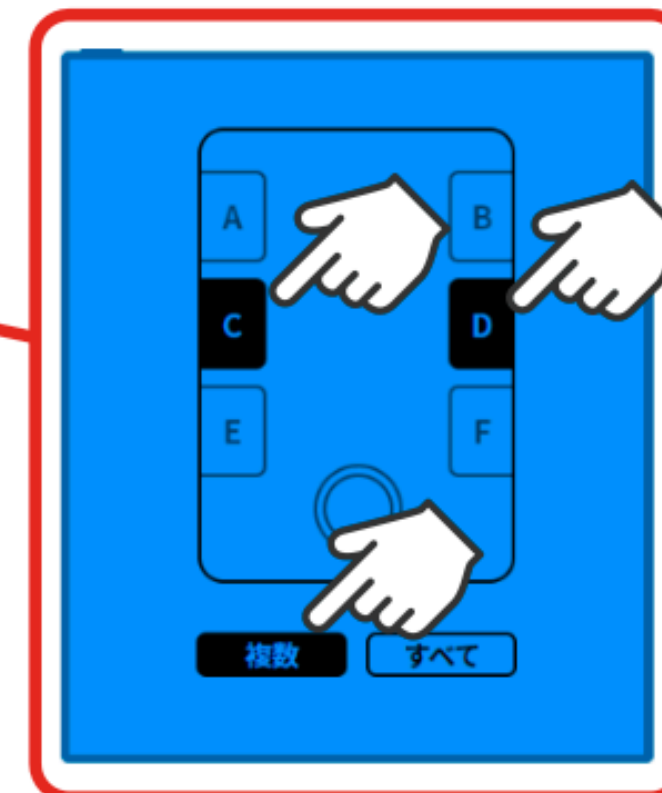


モータ 1 回転でロボットが
どれだけ前進するかを設定

「もし～なら」は「制御」にあります

「フォースセンサの押された」は「センサ」にあります

左右のモータを逆回転させるため、
あえてモータブロックを使用
「複数」をクリックして
ポート C とポート D を選択



フォースセンサによる障害物回避 (Python)



p.65 : force.py

```
from hub import port
import runloop
import motor_pair
import force_sensor
```

```
TURN_ANGLE = 90
VEL = 500
```

```
motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
```

```
4 async def backward(back_cm): #後退
    move_angle = int(back_cm*20.57)
    await motor_pair.move_tank_for_degrees(motor_pair.PAIR_1,move_angle, -VEL, -VEL)
    motor_pair.stop(motor_pair.PAIR_1)
```

続< →

フォースセンサによる障害物回避 (Python)



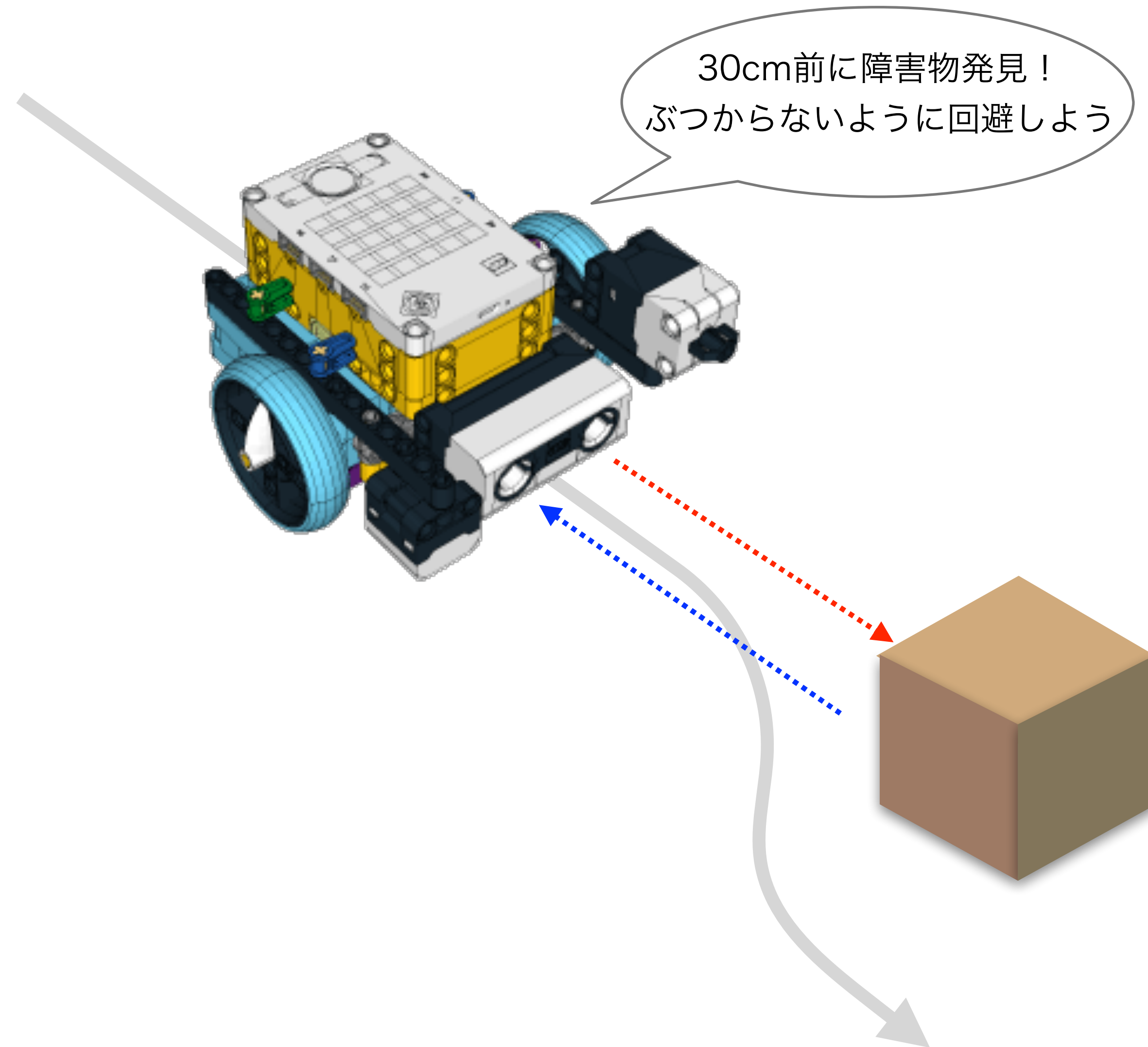
続き →

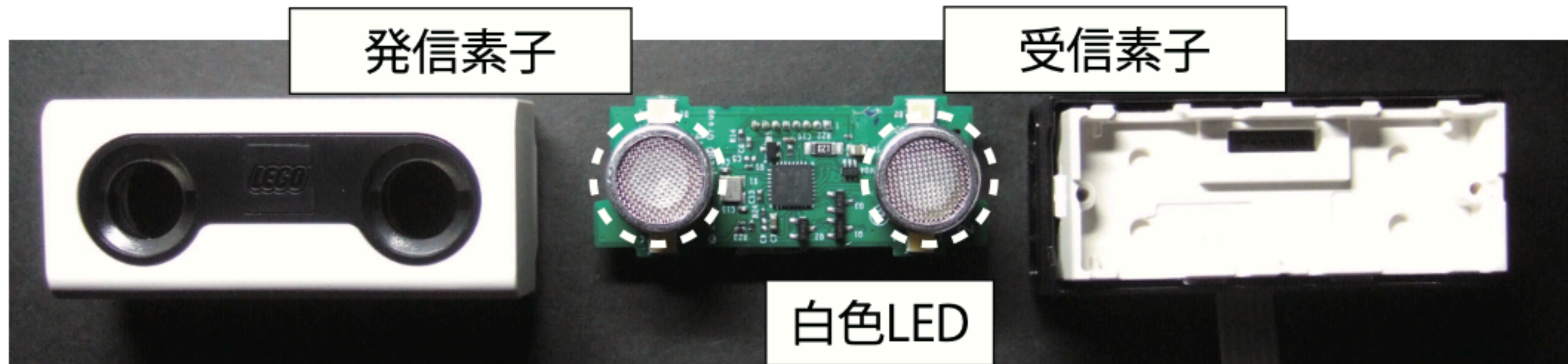
```
5 async def turn_right(turn_angle): #右旋回
    await motor_pair.move_tank_for_degrees(motor_pair.PAIR_1, turn_angle, VEL, -VEL)
    motor_pair.stop(motor_pair.PAIR_1)

async def main():
1 while True:
2     motor_pair.move_tank(motor_pair.PAIR_1, VEL, VEL)
3     if force_sensor.pressed(port.A):
4         await backward(10)
5         await turn_right(TURN_ANGLE)

runloop.run(main())
```

■障害物回避(距離センサ)

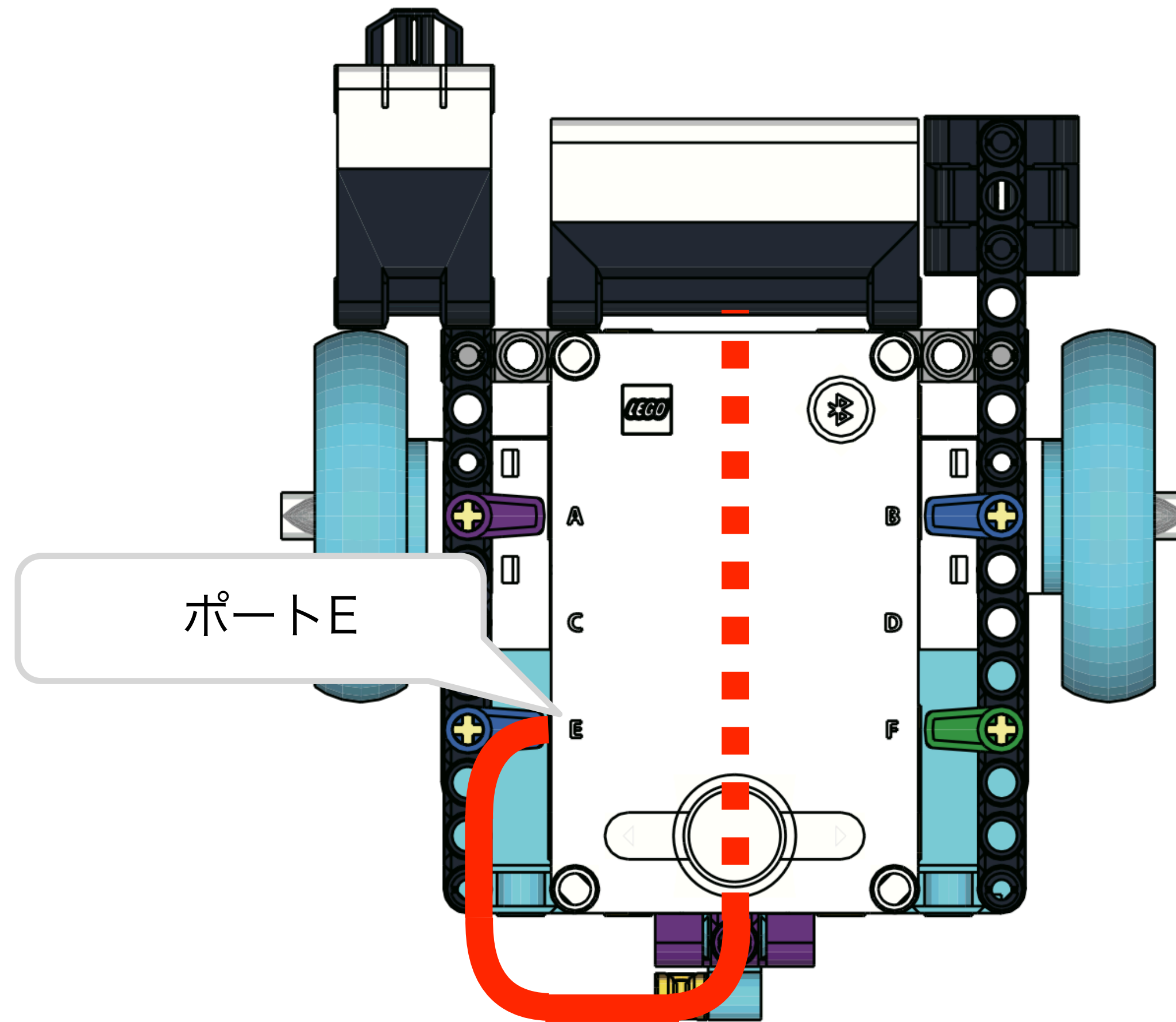




距離センサ（超音波センサ）の測定原理

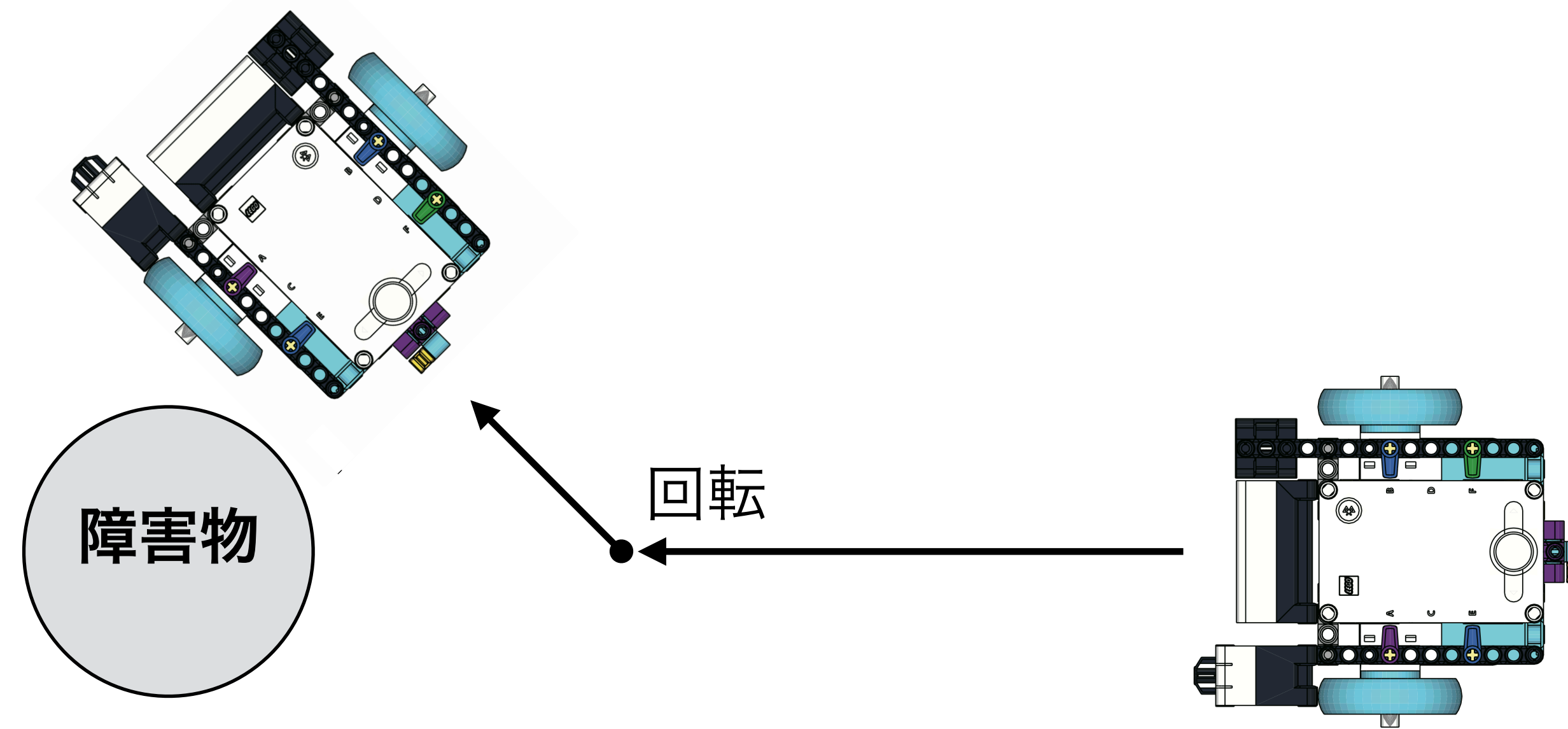
超音波を発信し、対象物で反射した超音波を受信し、この音波の発信から受信までの時間を計測することで対象物までの距離を計測

- ・ ポートEに距離センサを接続



※SPIKEではポートにどのセンサを接続しているか判定しているため、必ずポートEである必要はない

1. 常にロボットを前進 →無限ループの利用
2. 障害物までの距離を計測
3. $\left\{ \begin{array}{l} 30\text{cm以下(障害物あり)} : \text{右旋回. その後1.に戻る} \\ 30\text{cm以上(障害物なし)} : 1.\text{に戻る} \end{array} \right.$

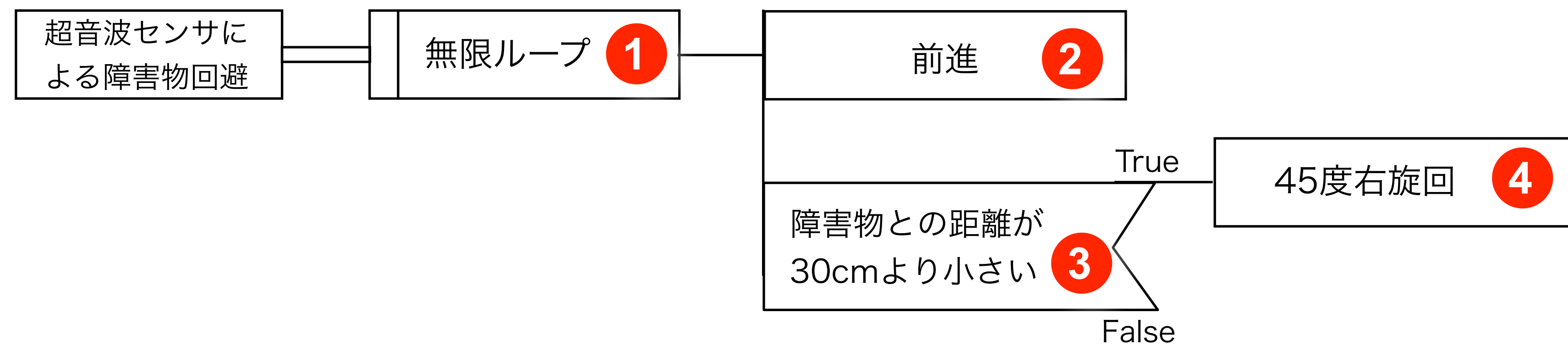


→ 後退する動作を必要としない

超音波センサによる障害物回避のPAD



- ・ 超音波センサの値により条件分岐して障害物回避



距離センサによる障害物回避



- 距離センサの値により、条件分岐して障害物回避

p.68 : usonic.llsp3

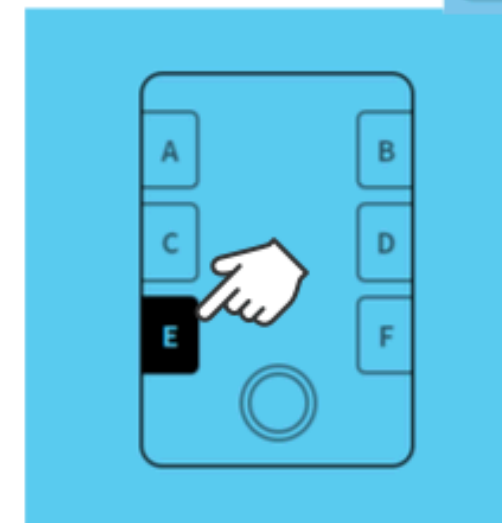


30cm より近い

30cm より近いなら

45度右旋回

センサー : 超音波センサ



ポート : E

距離センサによる障害物回避 (Python)



p.69 : usonic.py

```
from hub import port
import runloop
import motor_pair
import distance_sensor
```

```
TURN_ANGLE = 90
VEL = 500
```

```
motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
```

```
4 async def turn_right(turn_angle): #右旋回
    await motor_pair.move_tank_for_degrees(motor_pair.PAIR_1, turn_angle, VEL, -VEL)
    motor_pair.stop(motor_pair.PAIR_1)
```

```
async def main():
1   while True:
2       motor_pair.move_tank(motor_pair.PAIR_1, VEL, VEL)
3       if distance_sensor.distance(port.E) < 300 and \
           distance_sensor.distance(port.E) != -1:
4           await turn_right(TURN_ANGLE)
```

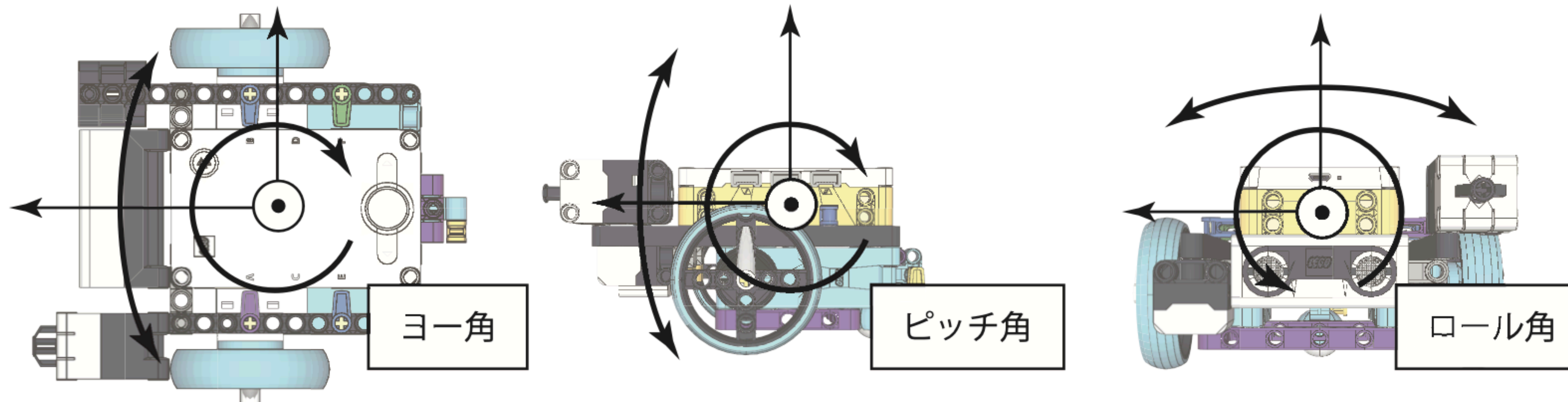
```
runloop.run(main())
```

■ モーションセンサを用いたモータ制御

モーションセンサ（ジャイロセンサ）の原理



- ・ SPIKEのモーションセンサは3軸(ヨー・ピッチ・ロール)の回転角/角速度を計測

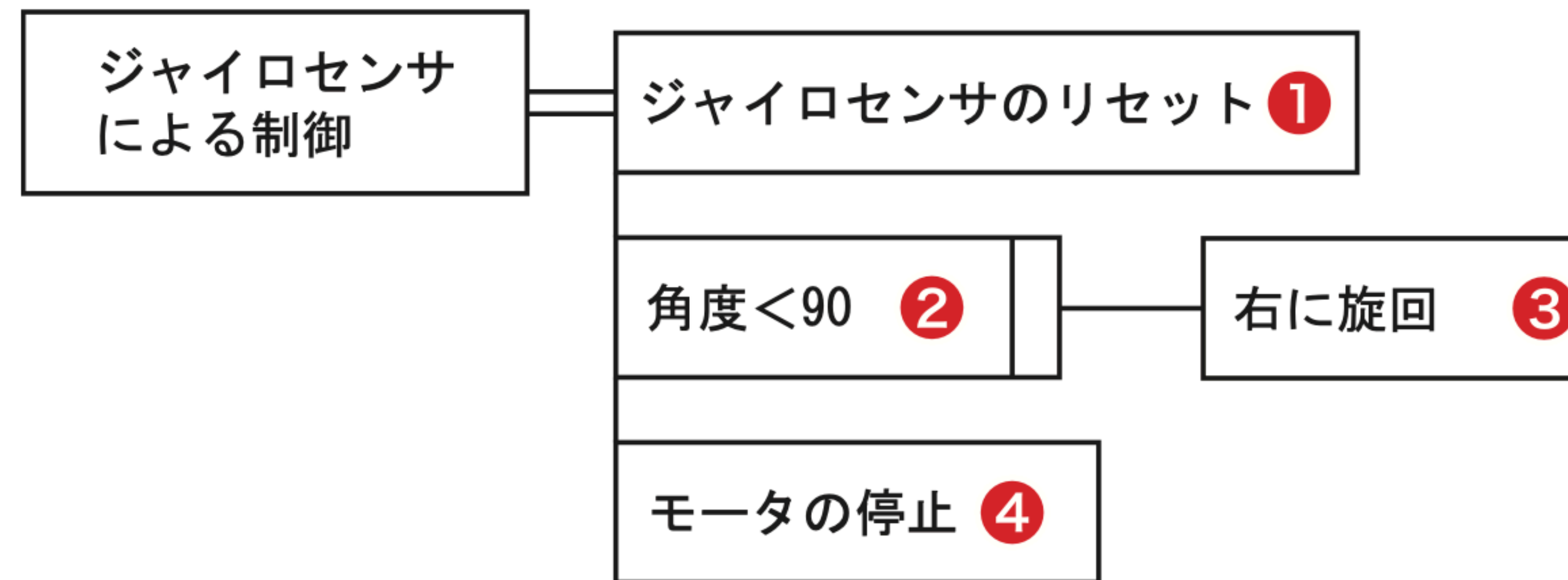


モーションセンサの測定原理

モーションセンサ(ジャイロセンサ)は角速度センサとも呼ばれ、単位時間あたりの回転角である角速度 [degree/s]を計測

簡単なアルゴリズム表現を使う

モーションセンサを使用して90度になるまでロボットを右に旋回→停止

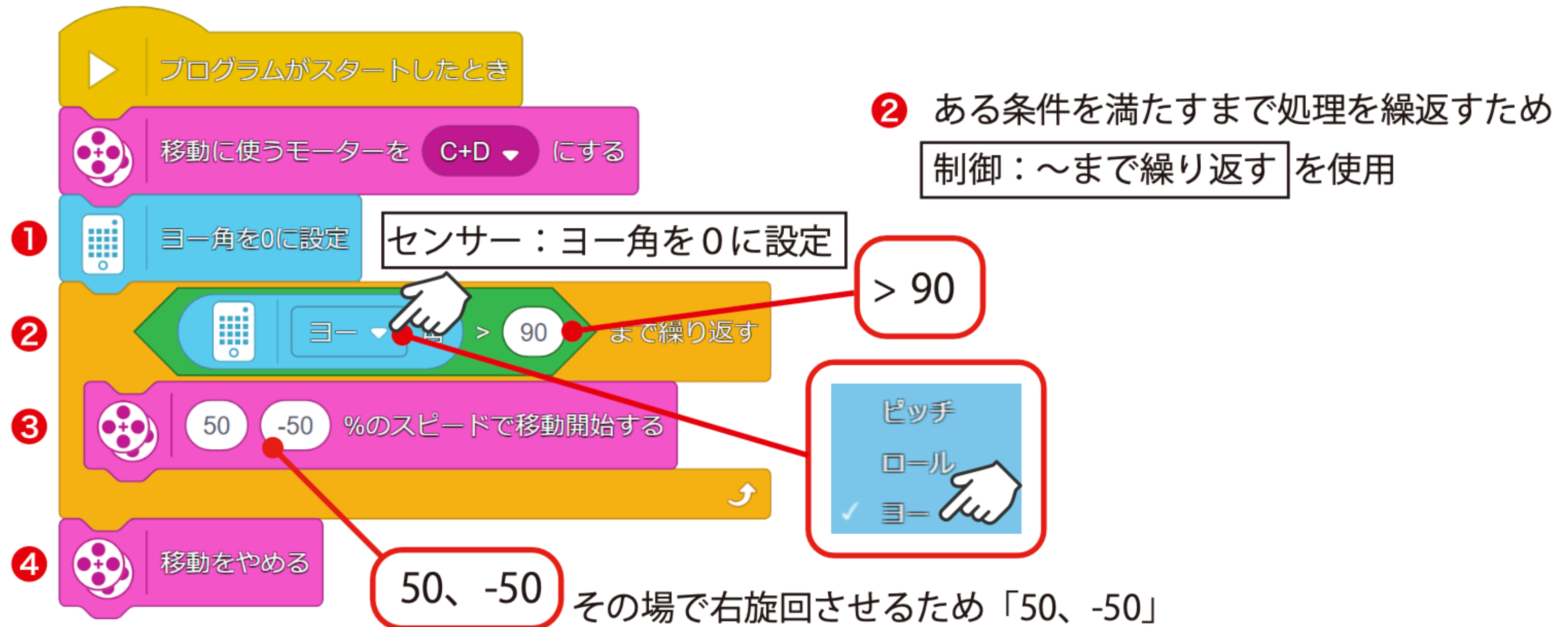


モーションセンサによる旋回角度制御



- モーションセンサの角度が90度以上になるまで右旋回

p.71 : gyro.llsp3



モーションセンサによる旋回角度制御 (Python)



- モーションセンサの角度が90度以上になるまで右旋回

p.72 : gyro.py

```
from hub import port
from hub import motion_sensor
import motor_pair
import runloop

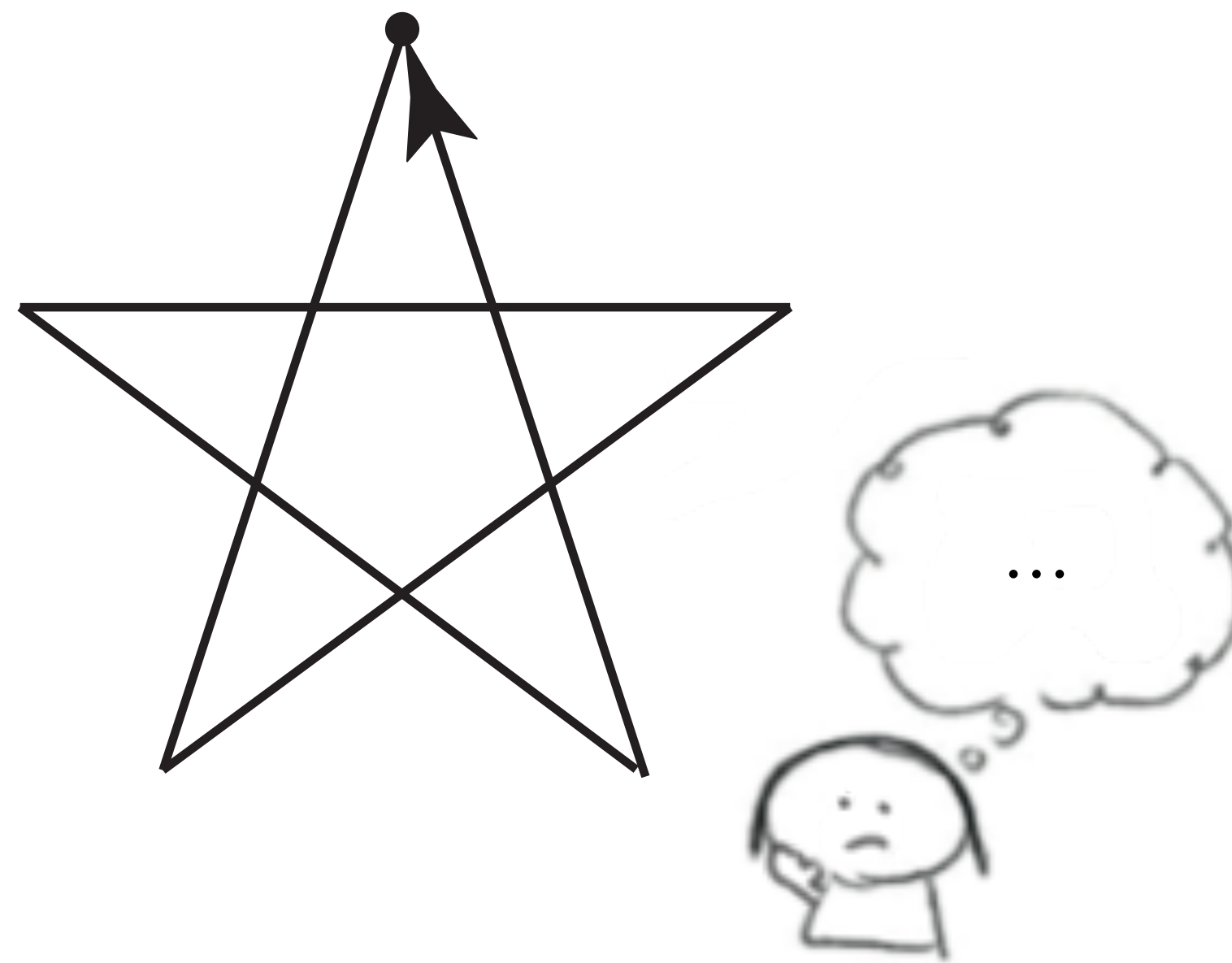
VEL = 100

async def main():
    motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)
    ① motion_sensor.set_yaw_face(motion_sensor.FRONT)
      motion_sensor.reset_yaw(0)

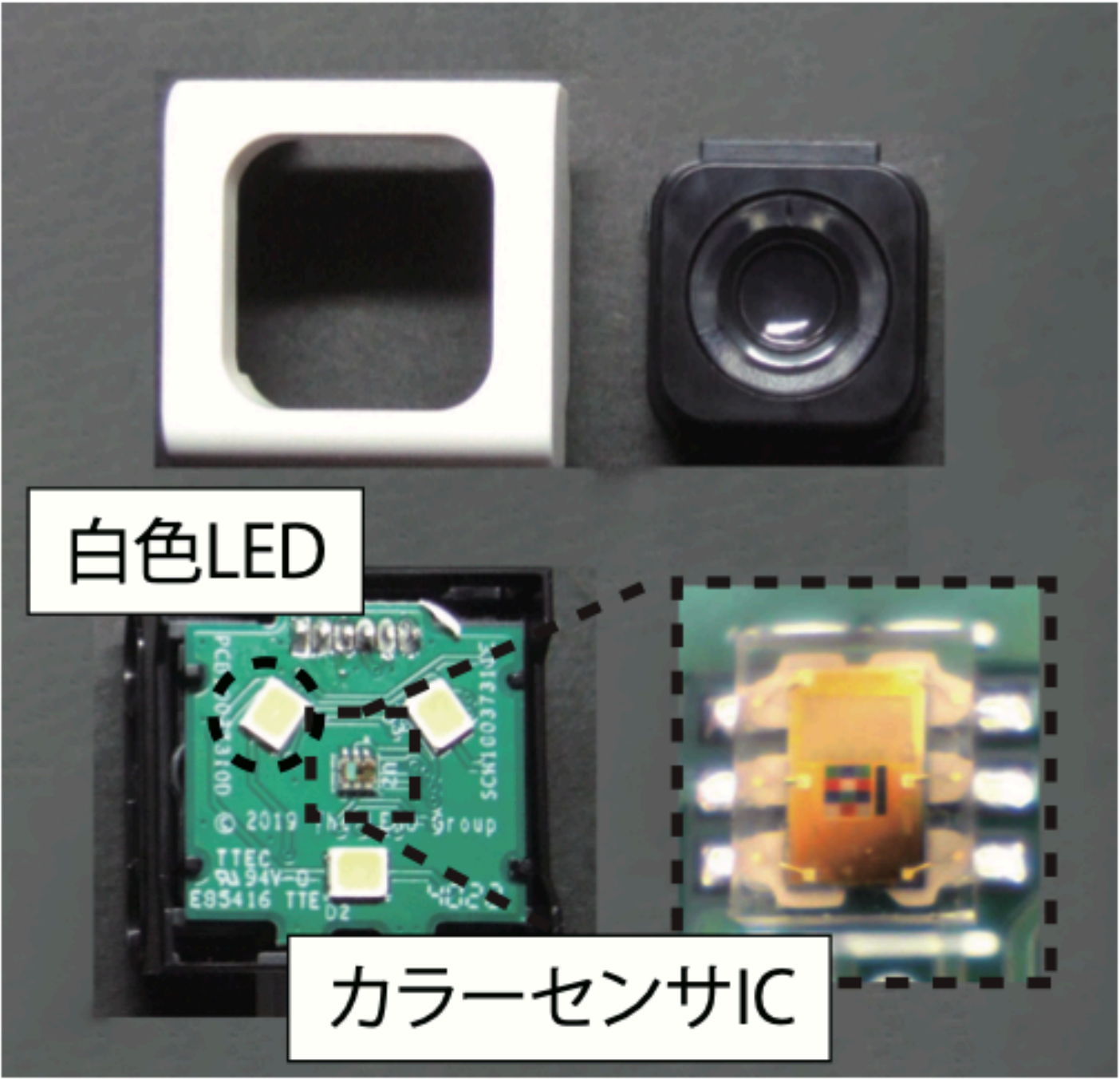
    ② while motion_sensor.tilt_angles()[0] > -900:
        ③ motor_pair.move_tank(motor_pair.PAIR_1, VEL, -VEL)
    ④ motor_pair.stop(motor_pair.PAIR_1)

runloop.run(main())
```

- ・ 基本問題
 - ・ モーションセンサを使用して45度右旋回して障害物回避を行うプログラムを作成する
- ・ 応用問題
 - ・ モーションセンサを使用して、下記のように動作するロボットプログラムを作成する



■カラーセンサによる色の認識



General Description

The TCS3472 device provides a digital return of red, green, blue (RGB), and clear light sensing values. An IR blocking filter, integrated on-chip and localized to the color sensing photodiodes, minimizes the IR spectral component of the incoming light and allows color measurements to be made accurately. The high sensitivity, wide dynamic range, and IR blocking filter make the TCS3472 an ideal color sensor solution for use under varying lighting conditions and through attenuating materials.

The TCS3472 color sensor has a wide range of applications including RGB LED backlight control, solid-state lighting, health/fitness products, industrial process controls and medical diagnostic equipment. In addition, the IR blocking filter enables the TCS3472 to perform ambient light sensing (ALS). Ambient light sensing is widely used in display-based products such as cell phones, notebooks, and TVs to sense the lighting environment and enable automatic display brightness for optimal viewing and power savings. The TCS3472, itself, can enter a lower-power wait state between light sensing measurements to further reduce the average power consumption.

Ordering Information and Content Guide appear at end of datasheet.

Key Benefits & Features

The benefits and features of TCS3472, Color Light-to-Digital Converter with IR Filter are listed below:

Benefits	Features
Enables accurate color and light sensing measurements under varying lighting conditions by minimizing IR and UV spectral component effects	- Red, Green, Blue (RGB), and Clear Light - Sensing with IR blocking filter - Programmable analog gain and integration time 3,800,000:1 dynamic range - Very high sensitivity - ideally suited for operation behind dark glass
Programmable interrupt pin enables level-style interrupts when pre-set values are exceeded, thus reducing companion micro-processor overhead	- Maskable interrupt - Programmable upper and lower thresholds with persistence filter

ams Datasheet
[v1-03] 2018-Mar-14

Page 1
Document Feedback

TCS3472 – Pin Assignment

Pin Assignment

The TCS3472 pin assignment is described below.

Figure 3:
Pin Diagram

Package FN Dual Flat No-Lead (Top View); Package drawing is not to scale.

Figure 4:
Pin Description

Pin Number	Pin Name	Pin Type	Description
1	V _{DD}		Supply voltage
2	SCL	Input	I ² C serial clock input terminal – clock signal for I ² C serial data
3	GND		Power supply ground. All voltages are referenced to GND
4	NC	Output	No connect - do not connect
5	INT	Output	Interrupt - open drain (active low).
6	SDA	Input/Output	I ² C serial data I/O terminal - serial data I/O for I ² C.

ams Datasheet
[v1-03] 2018-Mar-14

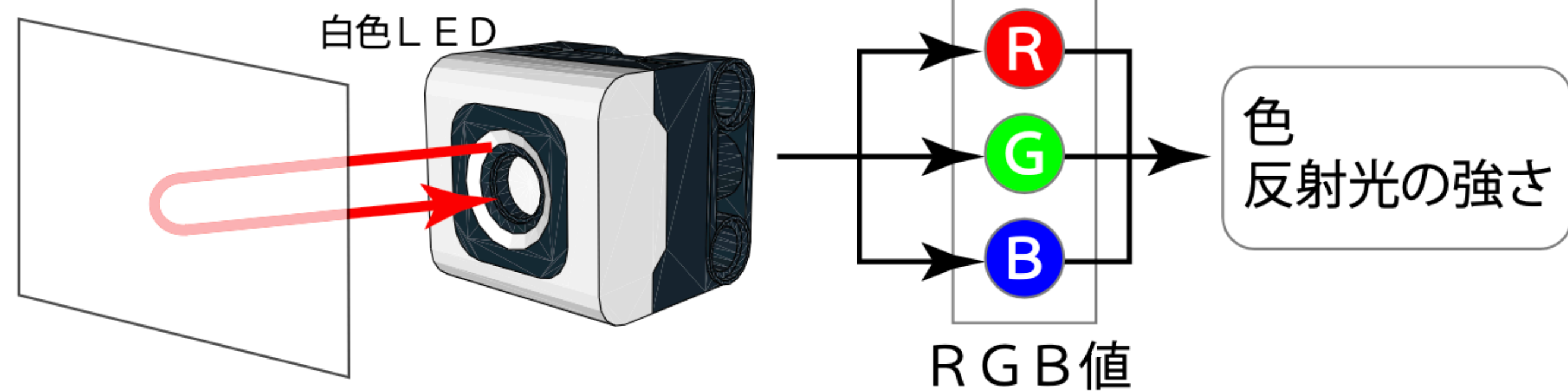
Page 5
Document Feedback

カラーセンサの測定原理

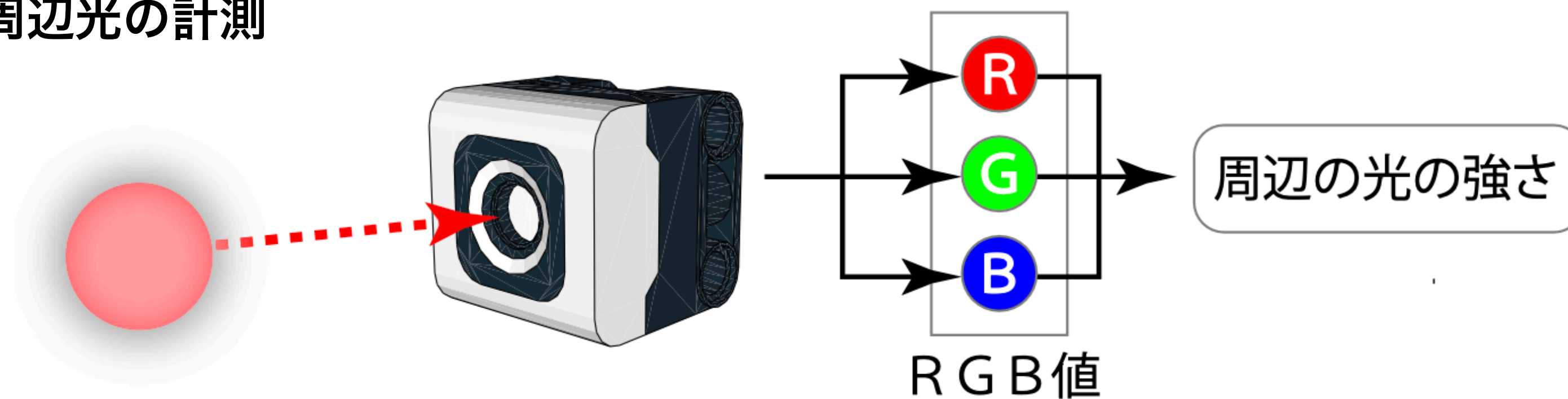
白色LEDにより光を照射して、それぞれの反射量(R, G, B成分)をICで読み取る

- 二種類のしくみで色や光の強さを計測

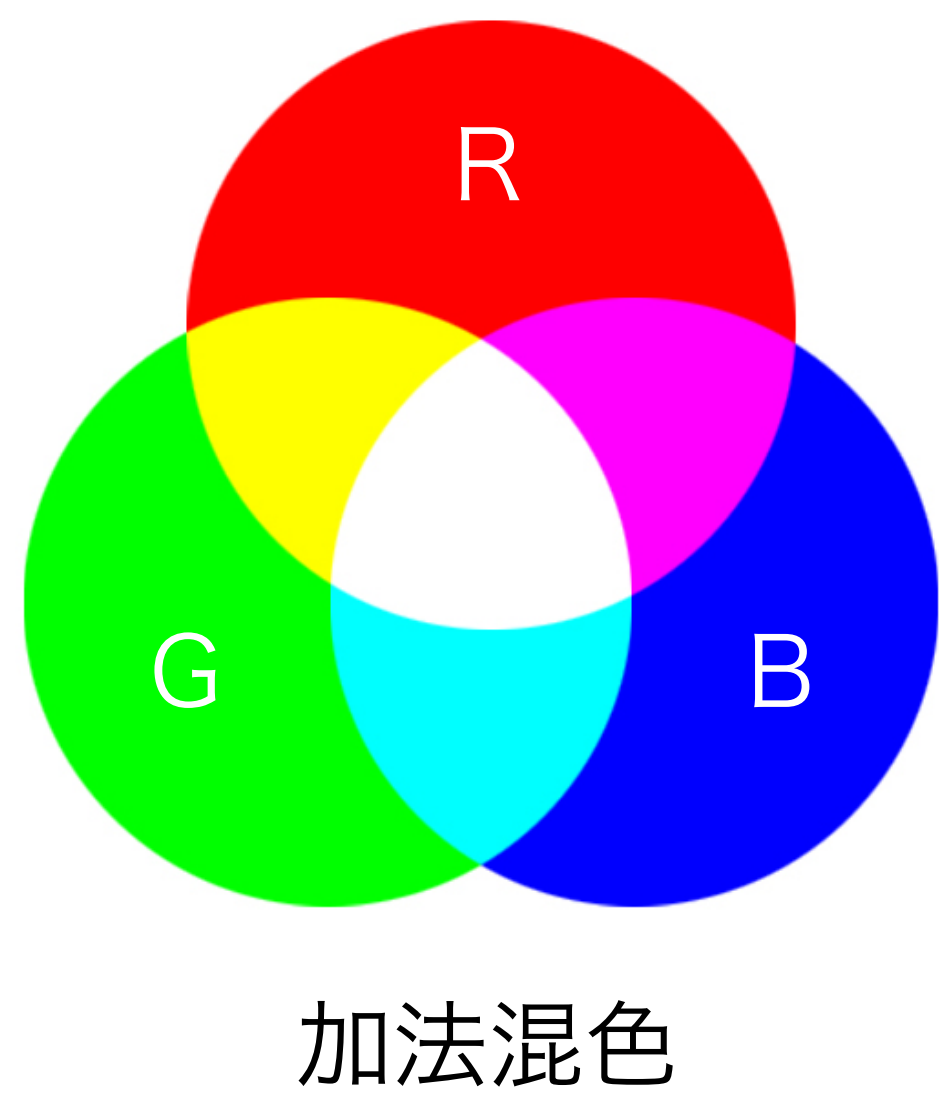
反射光の計測



周辺光の計測

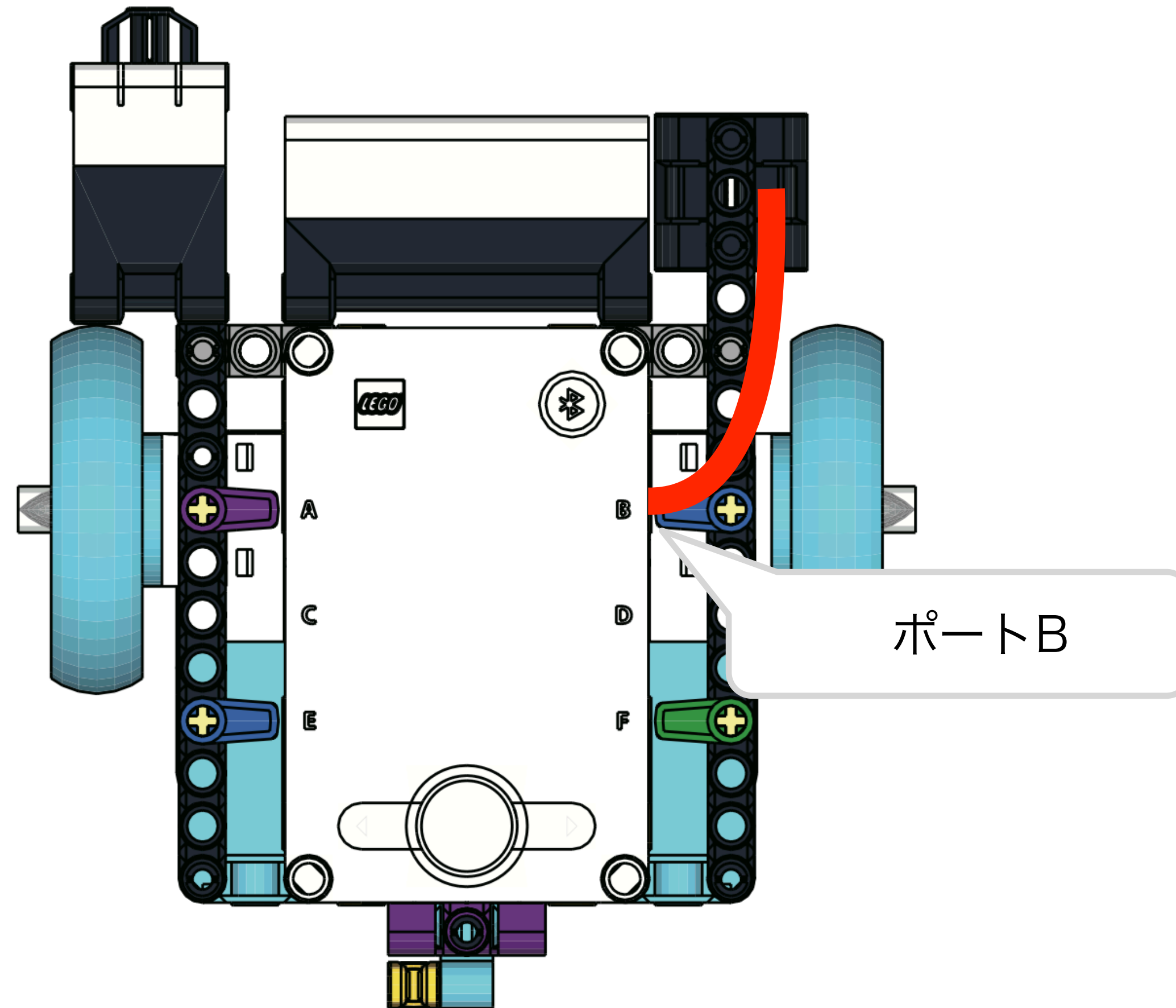


- 赤(R), 緑(G), 青(B)の3色を組み合わせせて色彩を表現



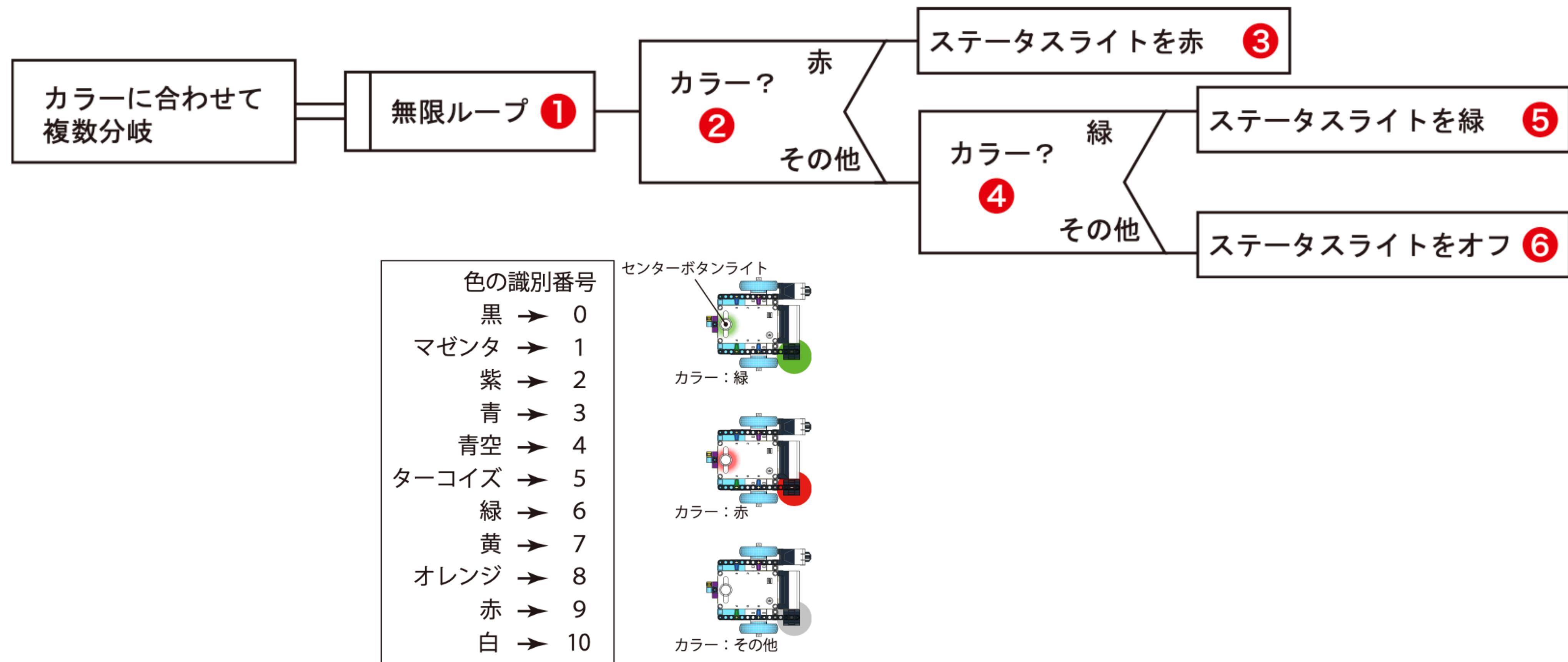
色番号	R	G	B
1: 黒	0	0	0
2: 青	0	0	大
3: 緑	0	大	0
4: 黄	大	大	0
5: 赤	大	0	0
6: 白	大	大	大

- ・ SPIKEのポートBにカラーセンサを接続



※SPIKEではポートにどのセンサを接続しているか判定しているため、必ずポートBである必要はない

- ・ 簡単なアルゴリズム表現を使う
 - カラーセンサで色を読み取り、センターボタンライトの色を変える

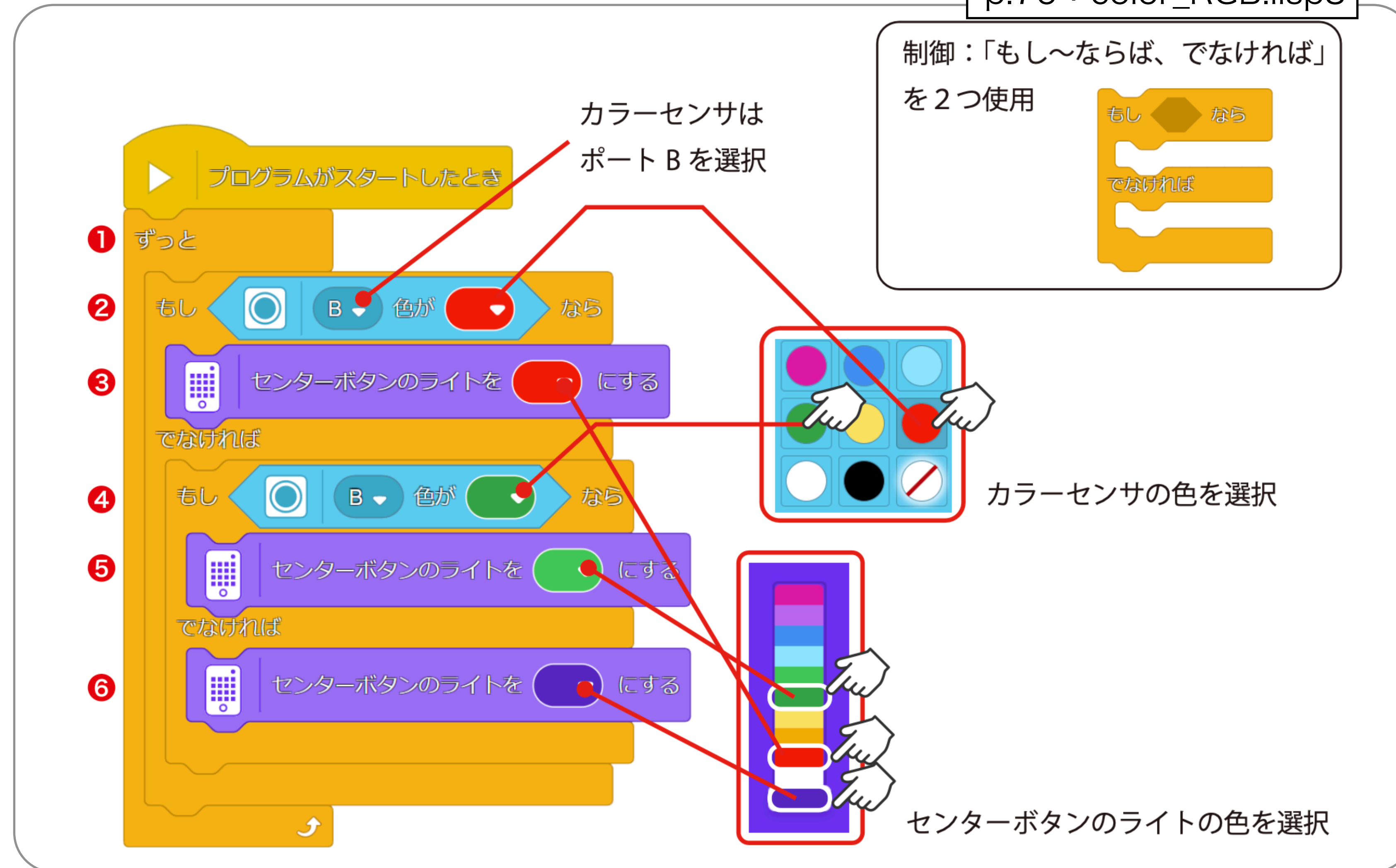


カラーセンサによる色の認識



- ・ カラーセンサの読み取りによってセンターボタンライトの色を変える

p.76 : color_RGB.llsp3



カラーセンサによる色の認識 (Python)



- ・ カラーセンサの読み取りによってセンターボタンライトの色を変える

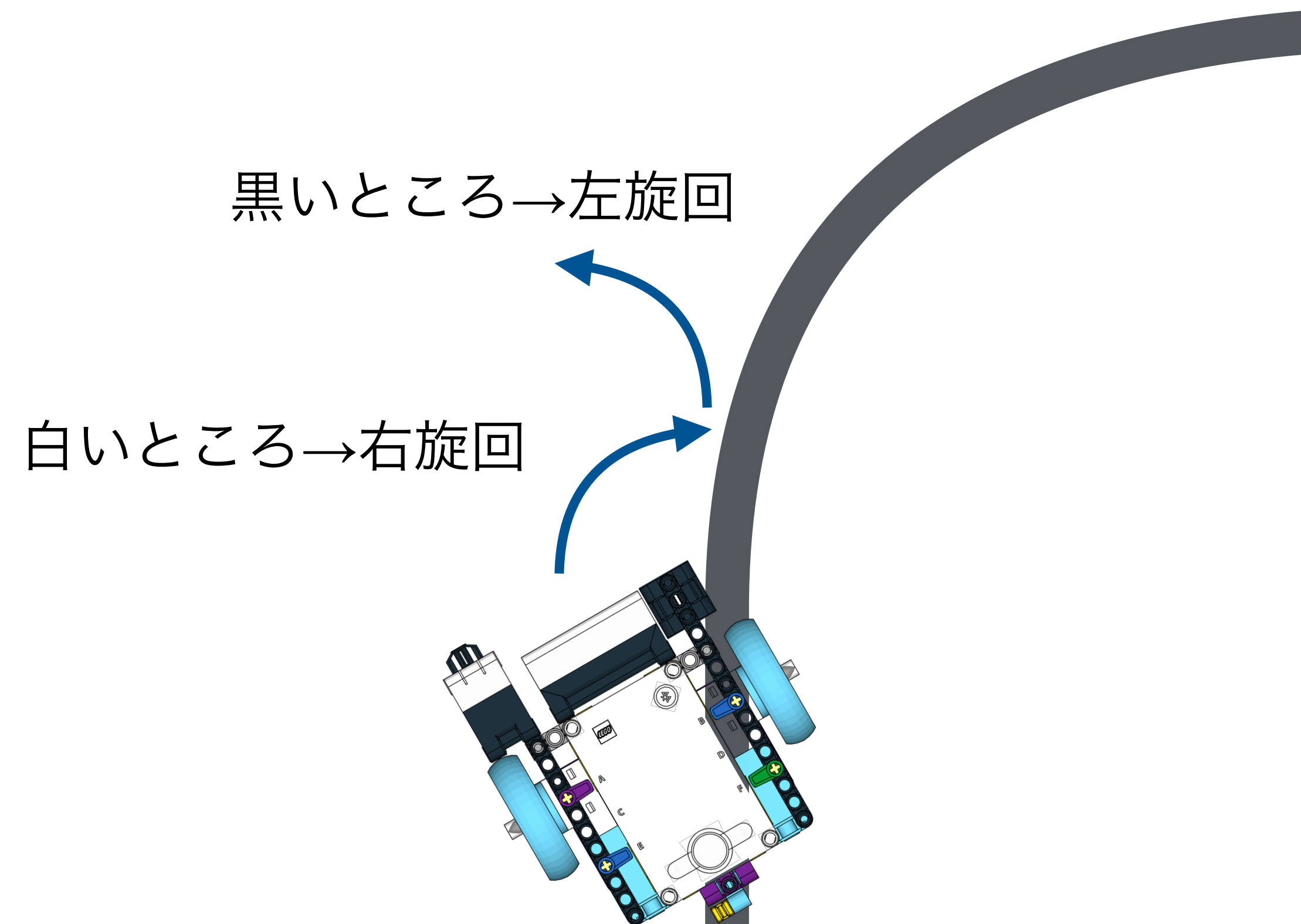
p.77 : color_RGB.py

```
from hub import port
from hub import light
import color_sensor
import color

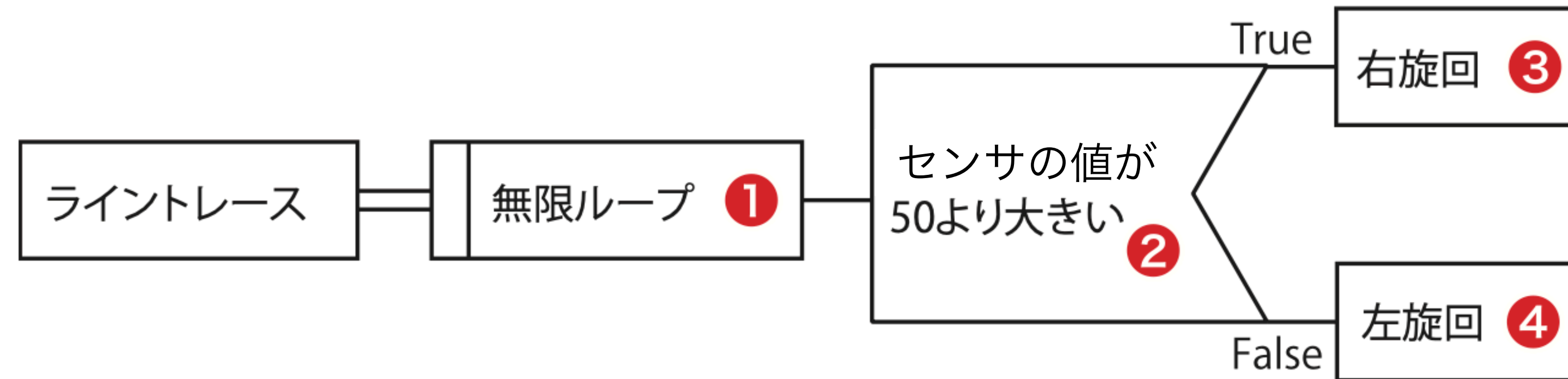
1 while True:
    2 if color_sensor.color(port.B) is color.RED:
        3 light.color(light.POWER, color.RED)
    4 elif color_sensor.color(port.B) is color.GREEN:
        5 light.color(light.POWER, color.GREEN)
    else:
        6 light.color(light.POWER, color.BLACK)
```


■カラーセンサによるライントレース

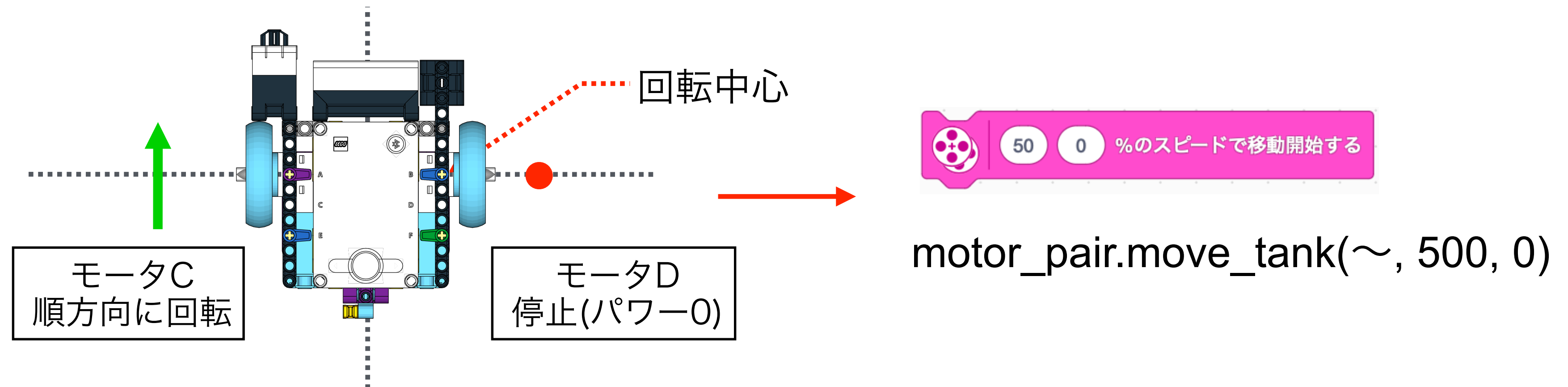
- ・ カラーセンサの”反射光の強さ”を利用して白と黒に判別
- ・ 白(明るい)ところでは右旋回, 黒(暗い)ところでは左旋回



- 白(明るい)ところでは右旋回, 黒(暗い)ところでは左旋回

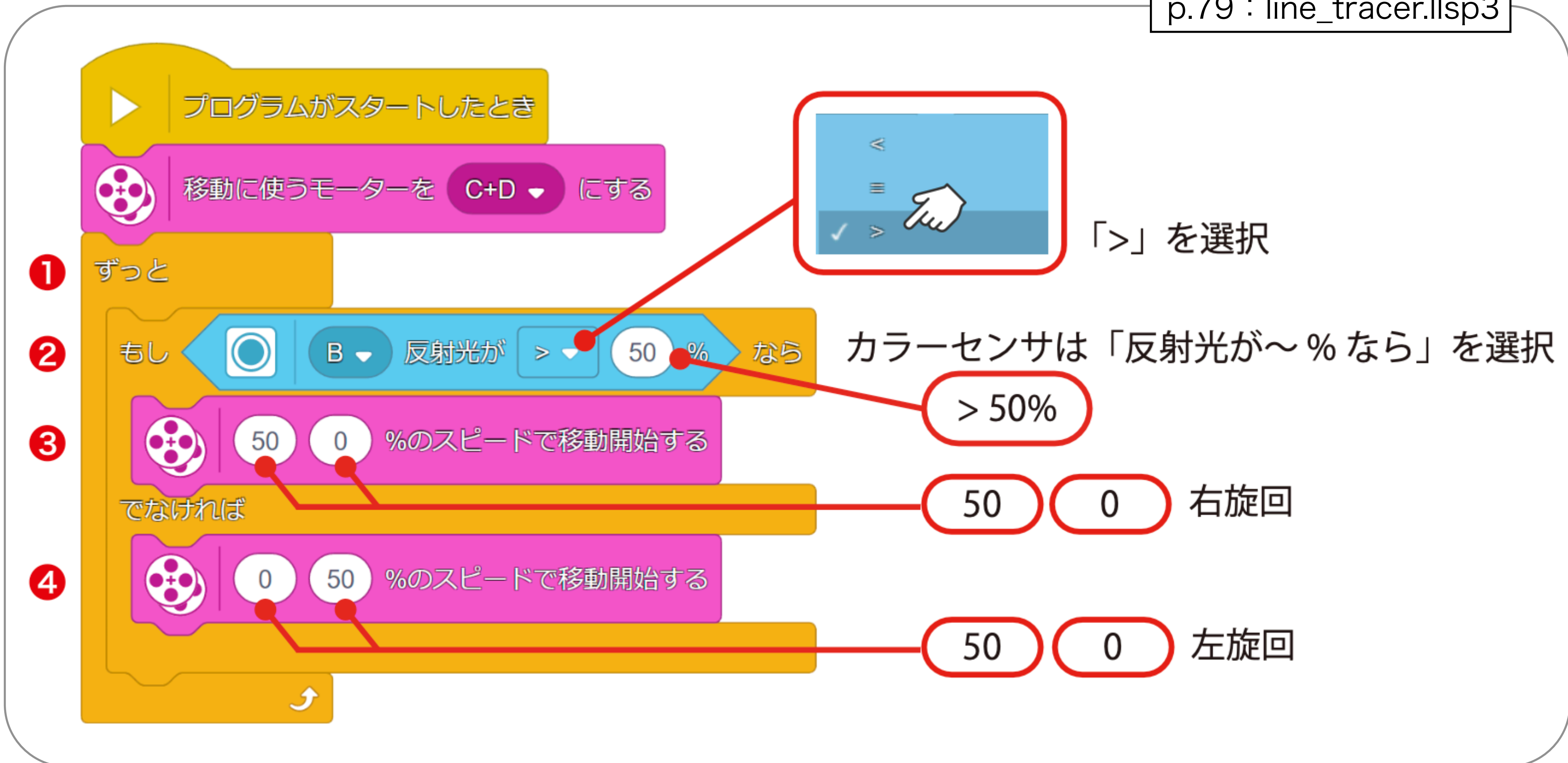


右回転しながら少し前に進むには：



- 白(明るい)ところでは右旋回, 黒(暗い)ところでは左旋回

p.79 : line_tracer.llsp3



- ・ 白(明るい)ところでは右旋回, 黒(暗い)ところでは左旋回

p.80 : line_tracer.py

```
from hub import port
import motor_pair
import color_sensor
import time
import runloop

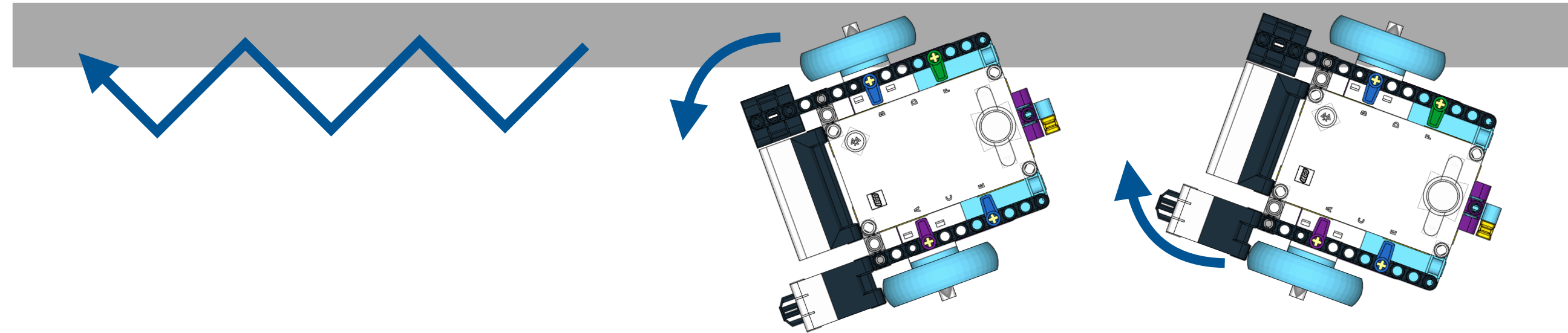
VEL = 500

motor_pair.pair(motor_pair.PAIR_1, port.C, port.D)

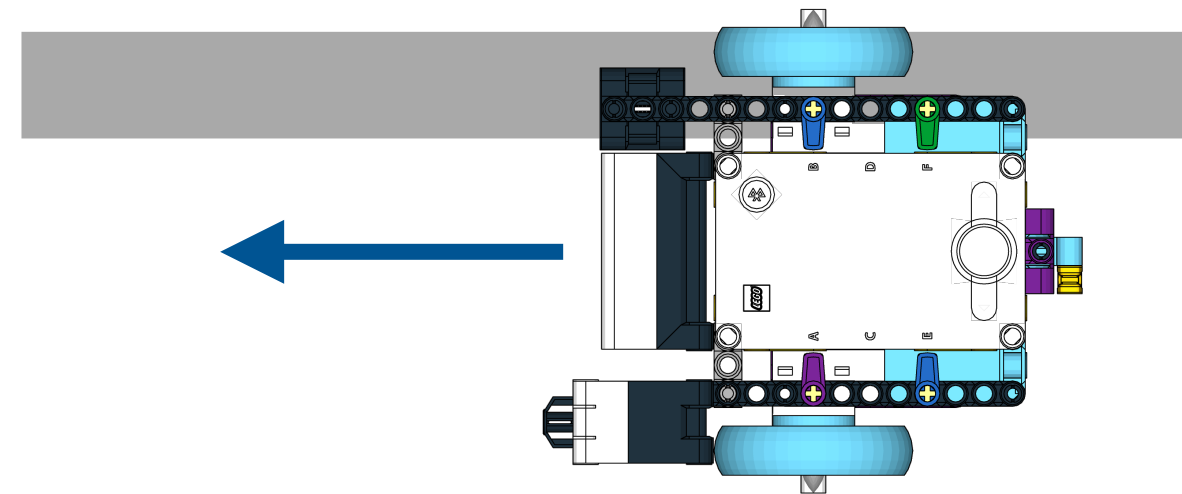
async def main():
    1 while True:
        2 if color_sensor.reflection(port.B) > 50:
            3 motor_pair.move_tank(motor_pair.PAIR_1, VEL, 0)
        else:
            4 motor_pair.move_tank(motor_pair.PAIR_1, 0, VEL)

runloop.run(main())
```

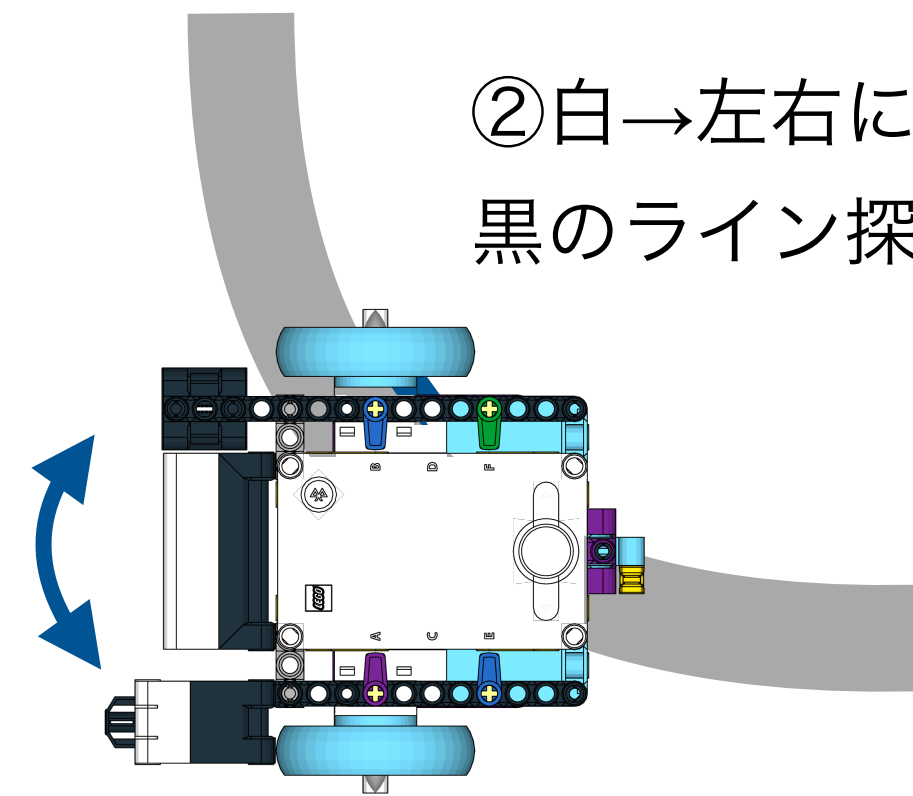

ジグザグ走行なので遅い！



①黒のライン上→前進



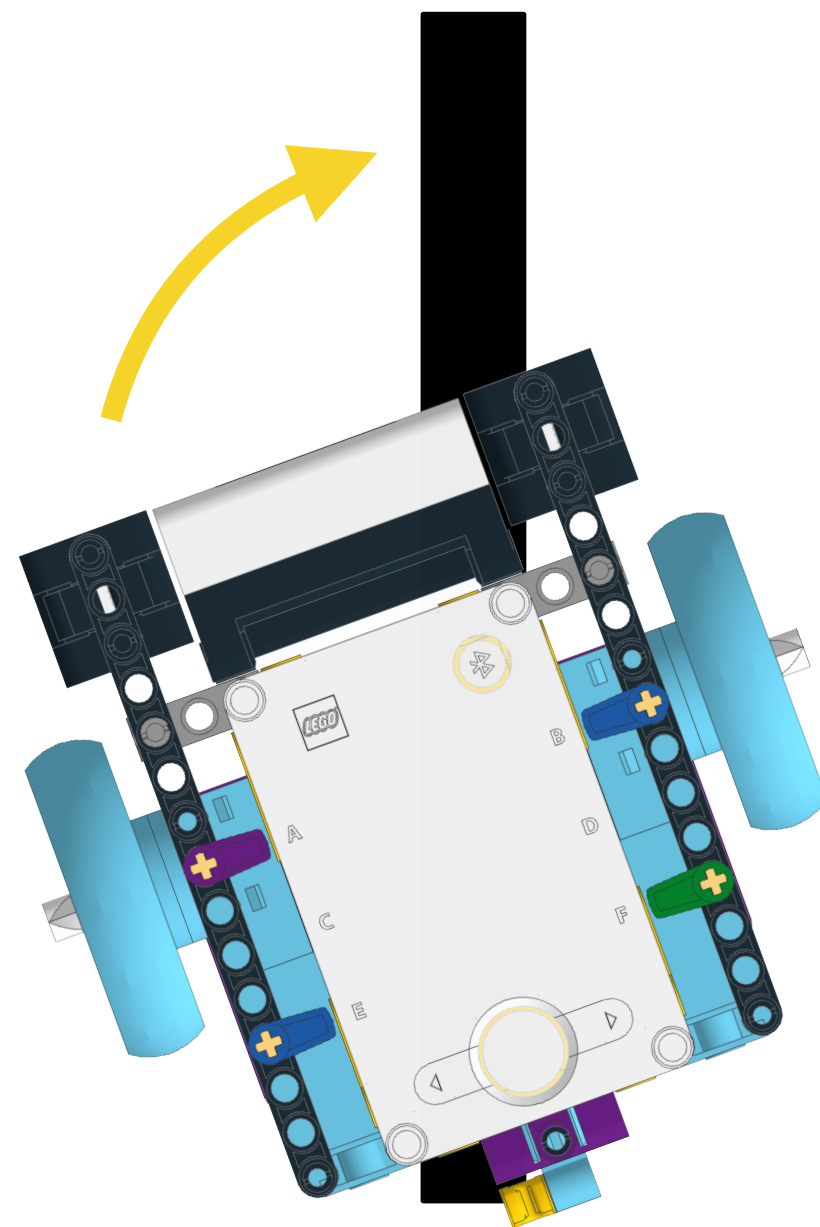
②白→左右に回転して
黒のライン探索



ライントレースの改良



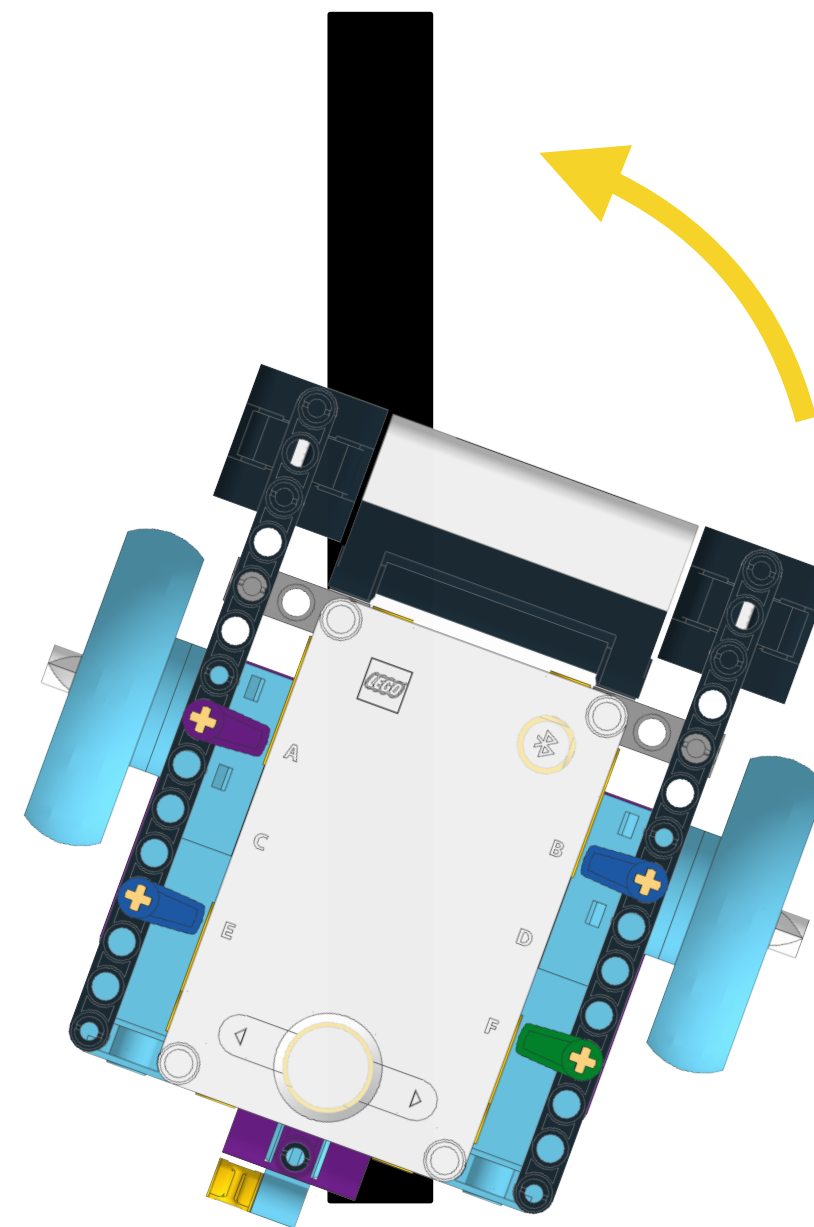
- ダブルカラーセンサによるライントレース
→ 左右のカラーセンサの読み取り値から次の動作を決定



白 & 黒



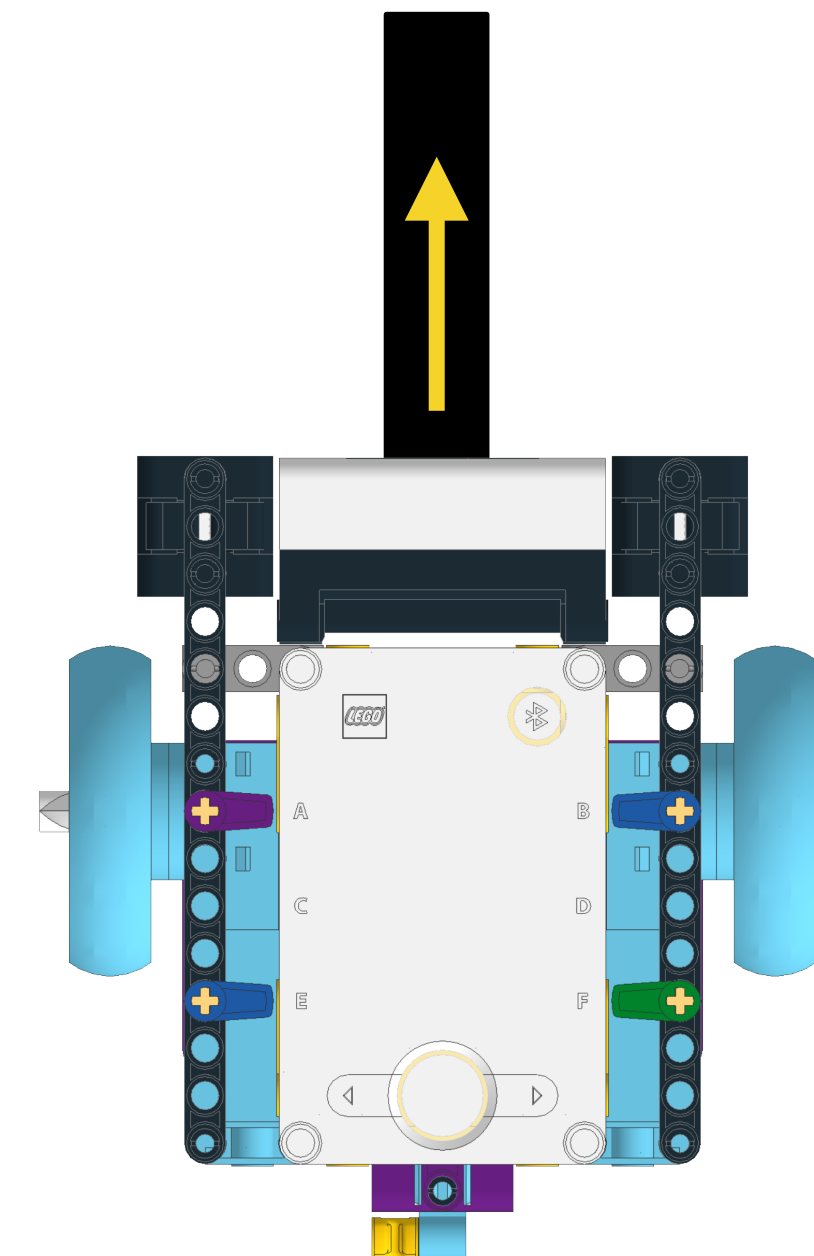
右旋回



黒 & 白



左旋回



(黒以外) & (黒以外)



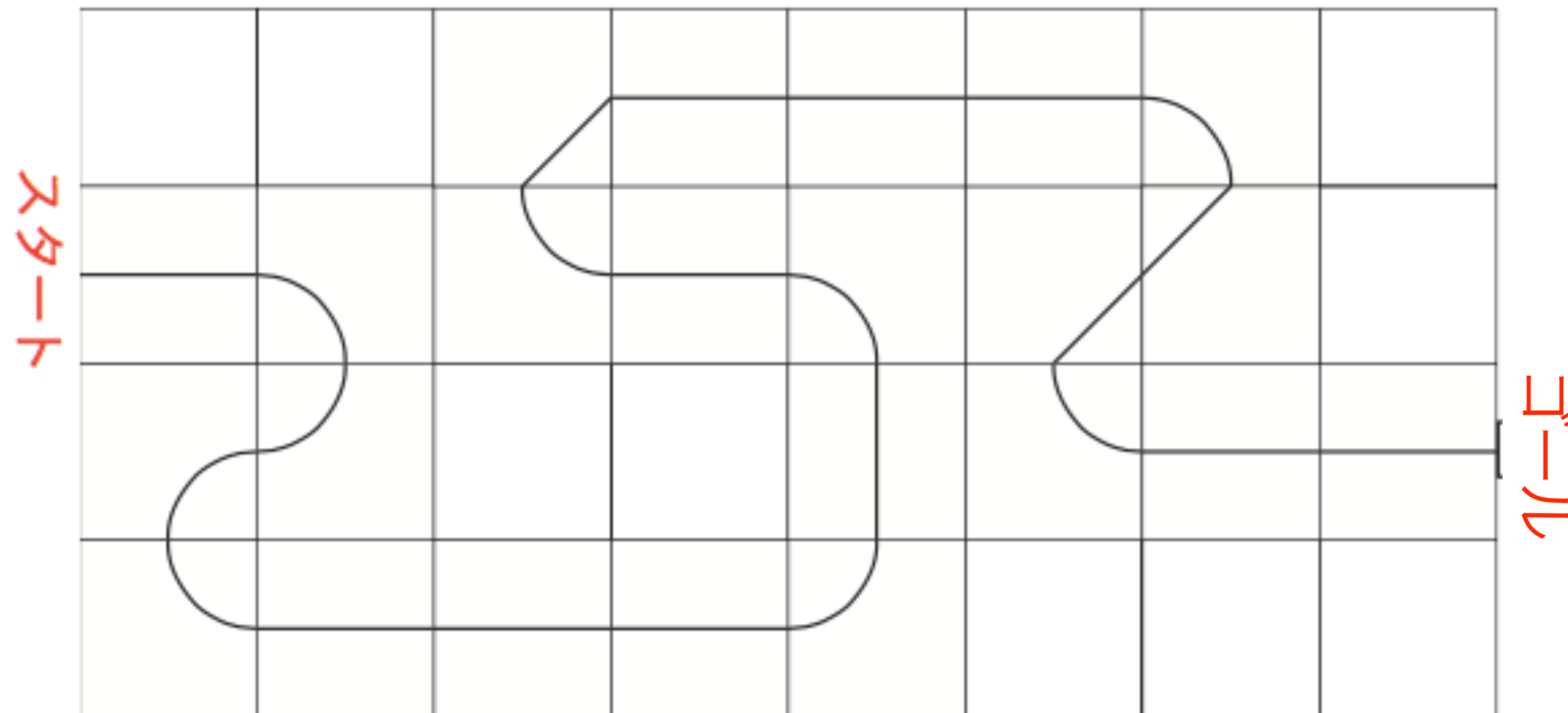
直進

■ 【演習課題】 ライトレース競技

【演習課題】 ライントレース競技

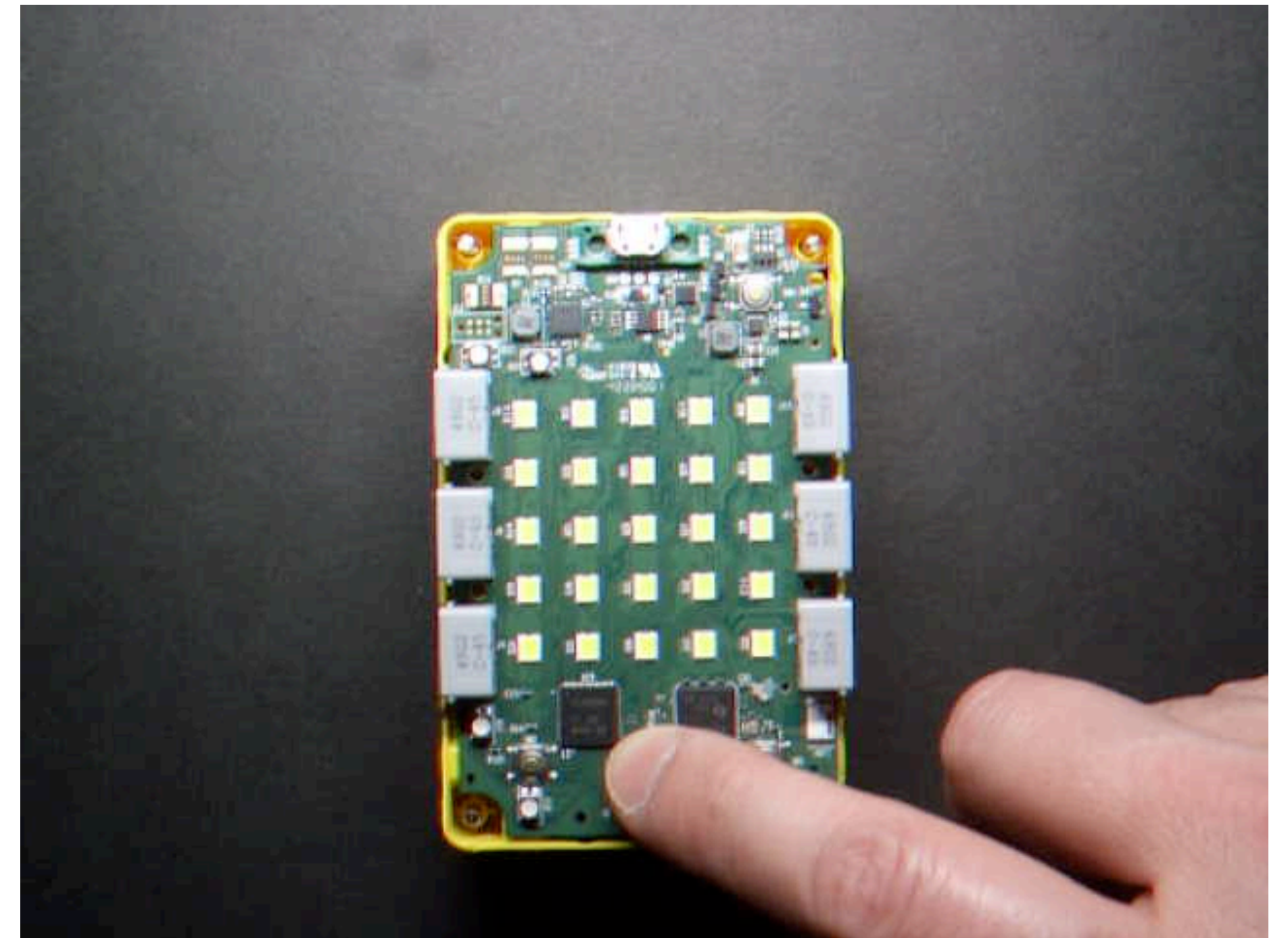
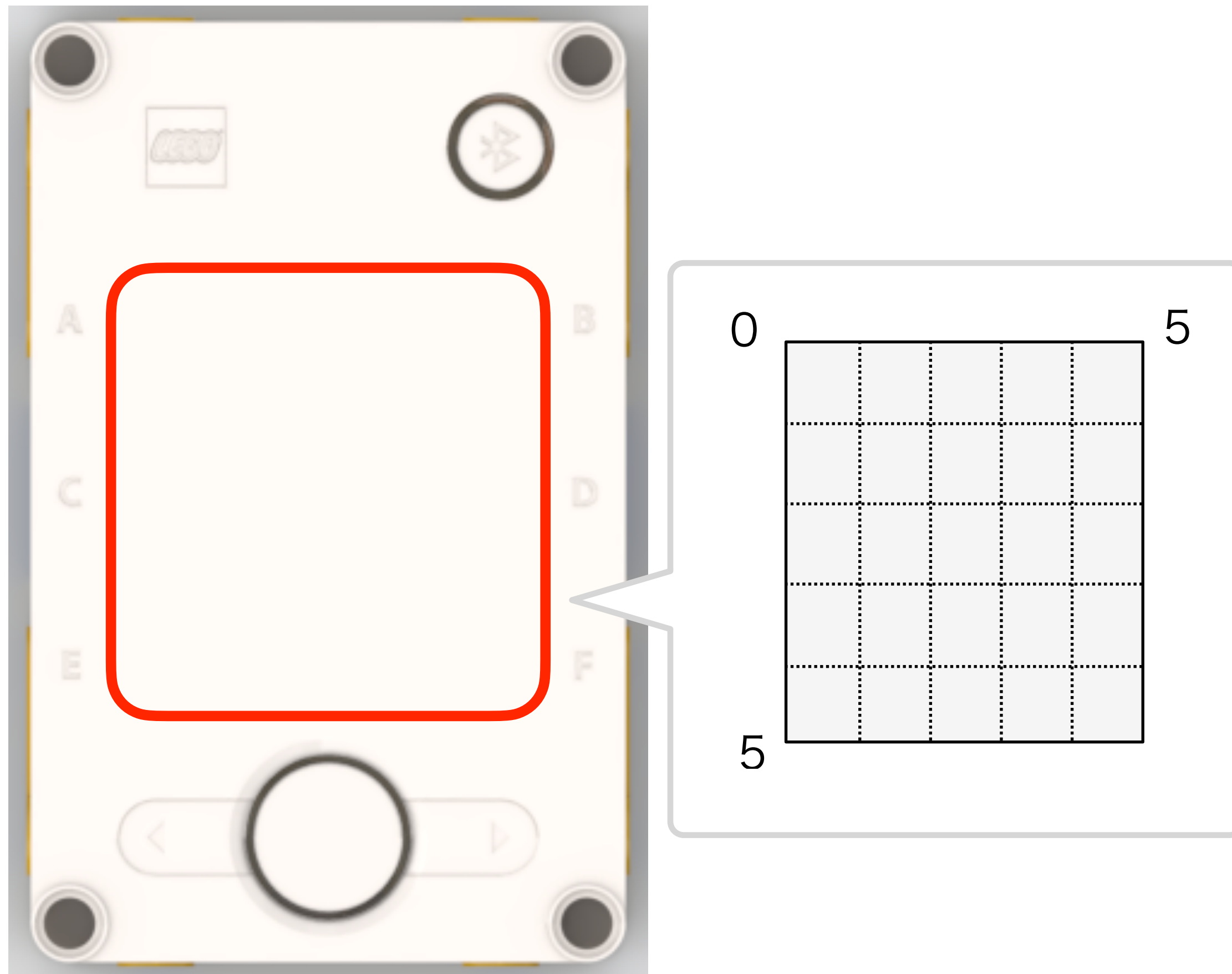


- ・ ライントレースをして早くゴールにたどり着いた人が勝ち
- ・ ライントレースしたパネル 1 枚通過で10点 (230点)
- ・ 同点の場合、ゴールまでの時間が短いチームが勝ち



■ディスプレイへの表示

- ・ SPIKE ハブの正面
 - 5x5 のLEDが内臓



距離センサの値によって表示を変更



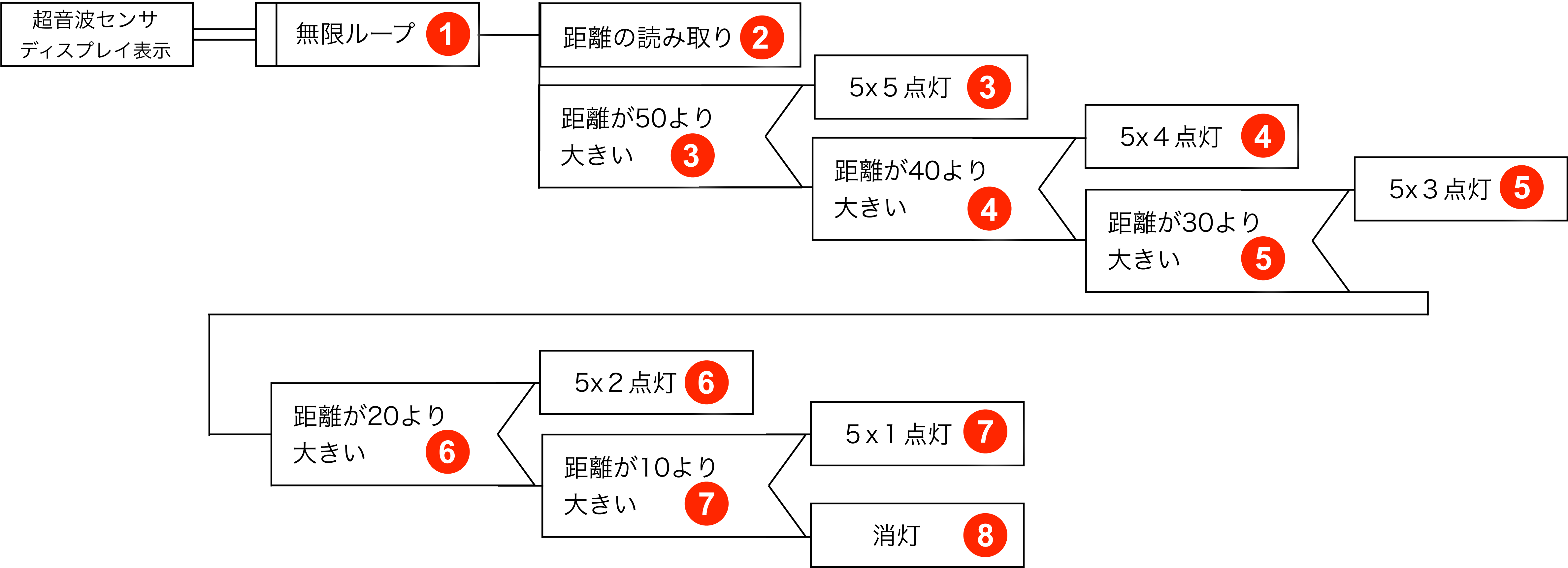
- ・ 距離センサの読み取り値によってLEDを変化させる
 - デバッグに重要



距離センサの値によってライトマトリックス変化 (Python)

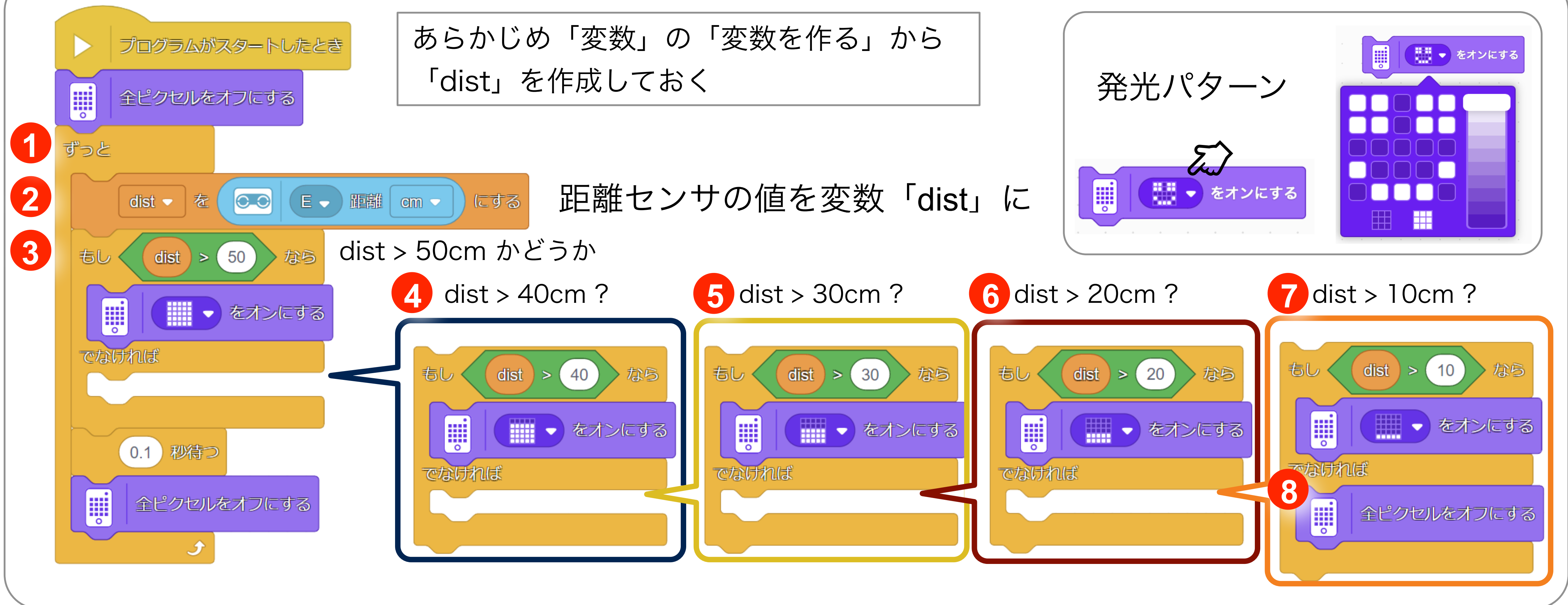


- 距離センサの値を読み取り、距離によってライトマトリックスを変化



- ・ 距離センサの値を読み取り、ライトマトリクスを変化

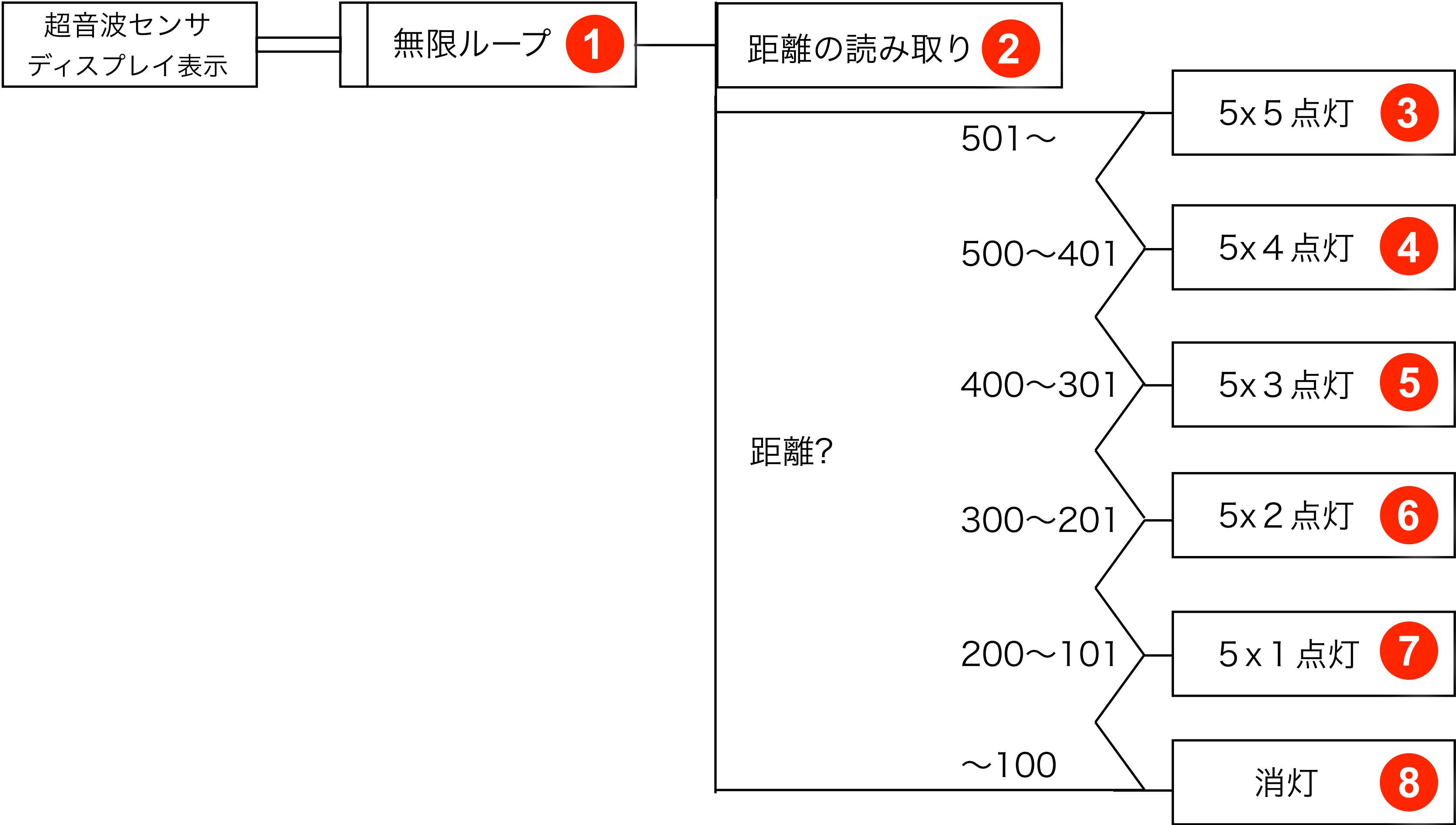
p.85 : display_dist.llsp3



距離センサの値によってライトマトリックス変化 (Python)



- 距離センサの値を読み取り、距離によってライトマトリックスを変化



距離センサの値によってライトマトリクス変化 (Python)



- 距離センサの値を読み取り、ライトマトリクスを変化

p.86 : display_dist.py

```
from hub import port
from hub import light_matrix
import runloop
import distance_sensor
import time

async def main():
    light_matrix.clear()
    1 while True:
        2 dist = distance_sensor.distance(port.E)
        3 if dist > 500:
            pixels = [100] * 25
        4 elif dist > 400:
            pixels1 = [0] * 5
            pixels2 = [100] * 20
            pixels = pixels1 + pixels2
```

続き →

```
5 elif dist > 300:
    pixels1 = [0] * 10
    pixels2 = [100] * 15
    pixels = pixels1 + pixels2
6 elif dist > 200:
    pixels1 = [0] * 15
    pixels2 = [100] * 10
    pixels = pixels1 + pixels2
7 elif dist > 100:
    pixels1 = [0] * 20
    pixels2 = [100] * 5
    pixels = pixels1 + pixels2
8 else:
    pixels = [0] * 25
    light_matrix.show(pixels)
    time.sleep_ms(100)
```

```
runloop.run(main())
```

続く →