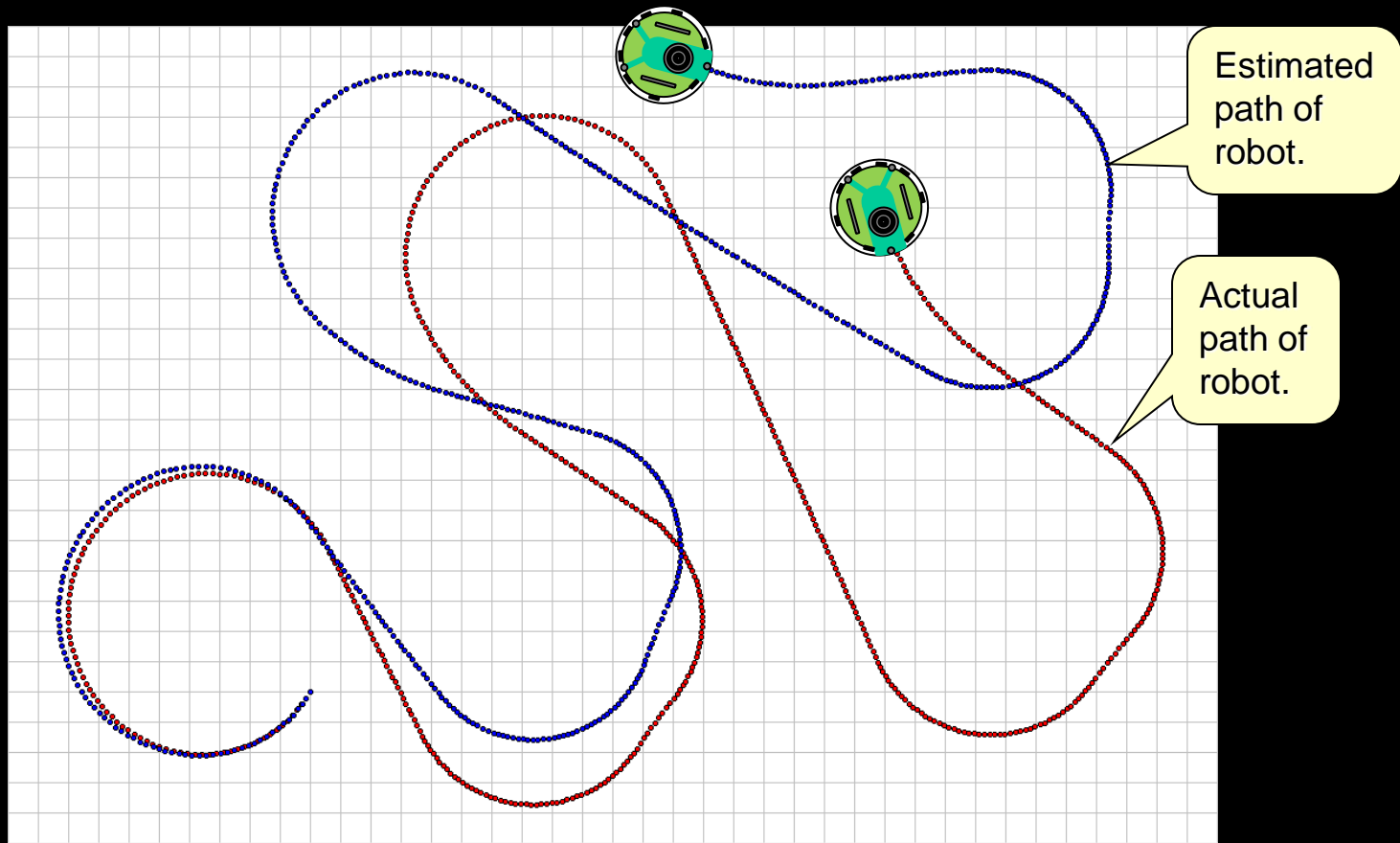


# Grid-Based Estimation

# Odometry Problems

- Using forward kinematics calculations, the odometry error is unbounded and will grow over time.

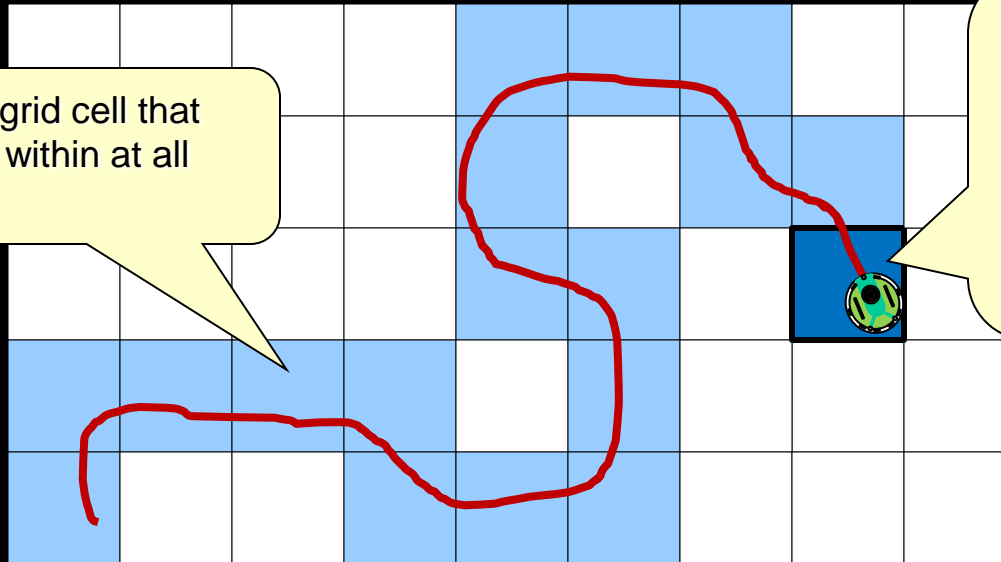


# Grid-Based Estimation

- Another position-estimation approach is to estimate the position of a robot according to its position within a fixed-size grid (just like a GPS grid cell).



Estimate grid cell that robot lies within at all times.

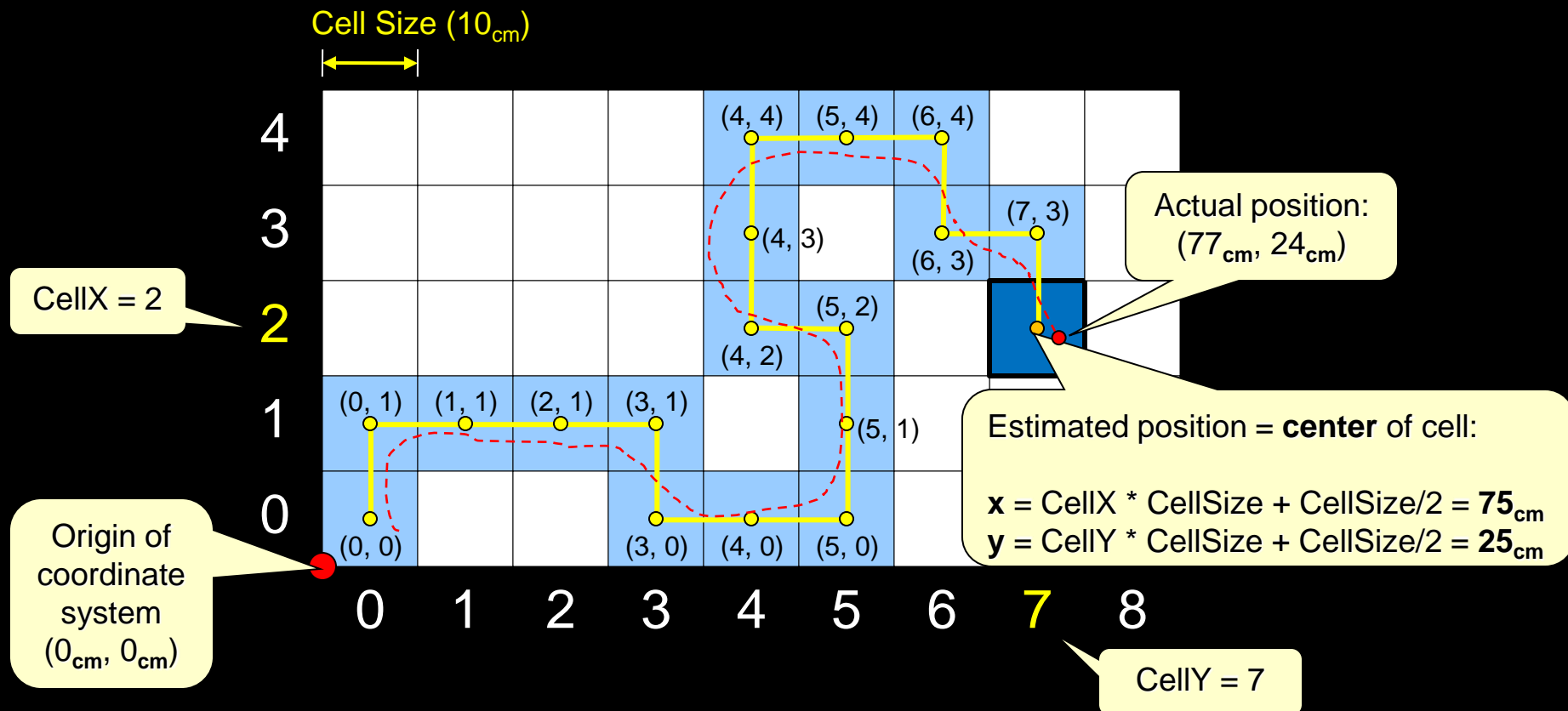


Don't know exact position at any time, but can say with reasonable certainty that robot is somewhere within a particular grid cell.

The position error is bounded ... it does not grow over time !

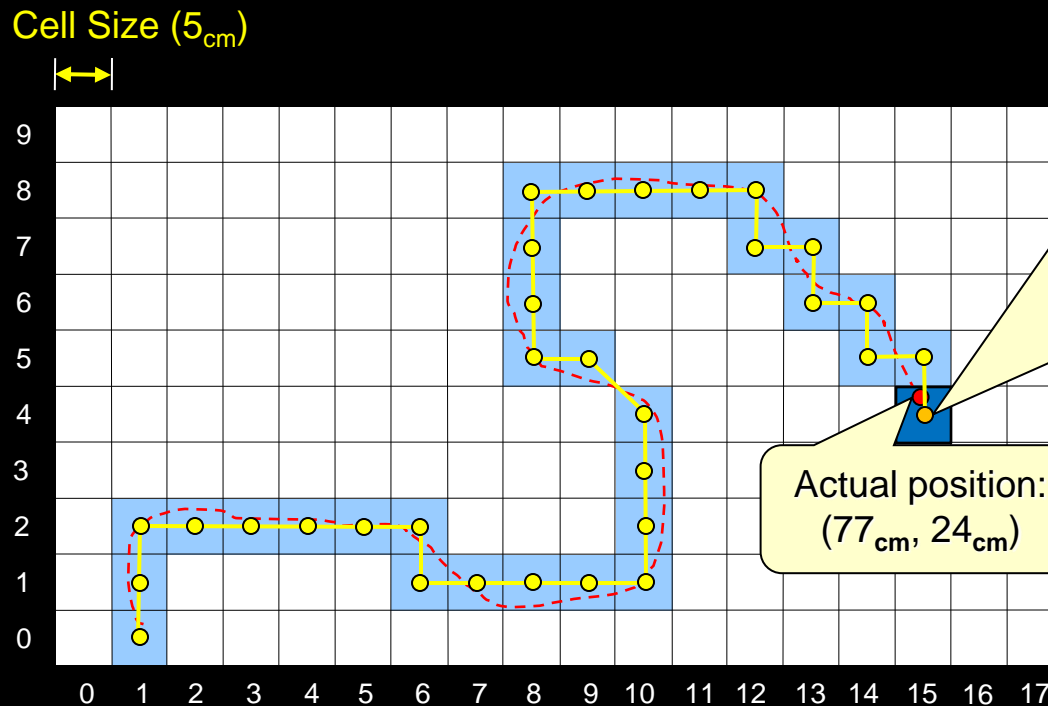
# Grid-Based Estimation

- Always have **rough idea** of robot location at any time.
  - Need to detect when robot crosses from one cell to another, then update cell number accordingly. Assume only local sensors available.



# Grid-Based Estimation

- A fine grid will produce a more accurate result.
- Path will be more “true” to actual robot path.



Compute position estimate to center of cell as follows:

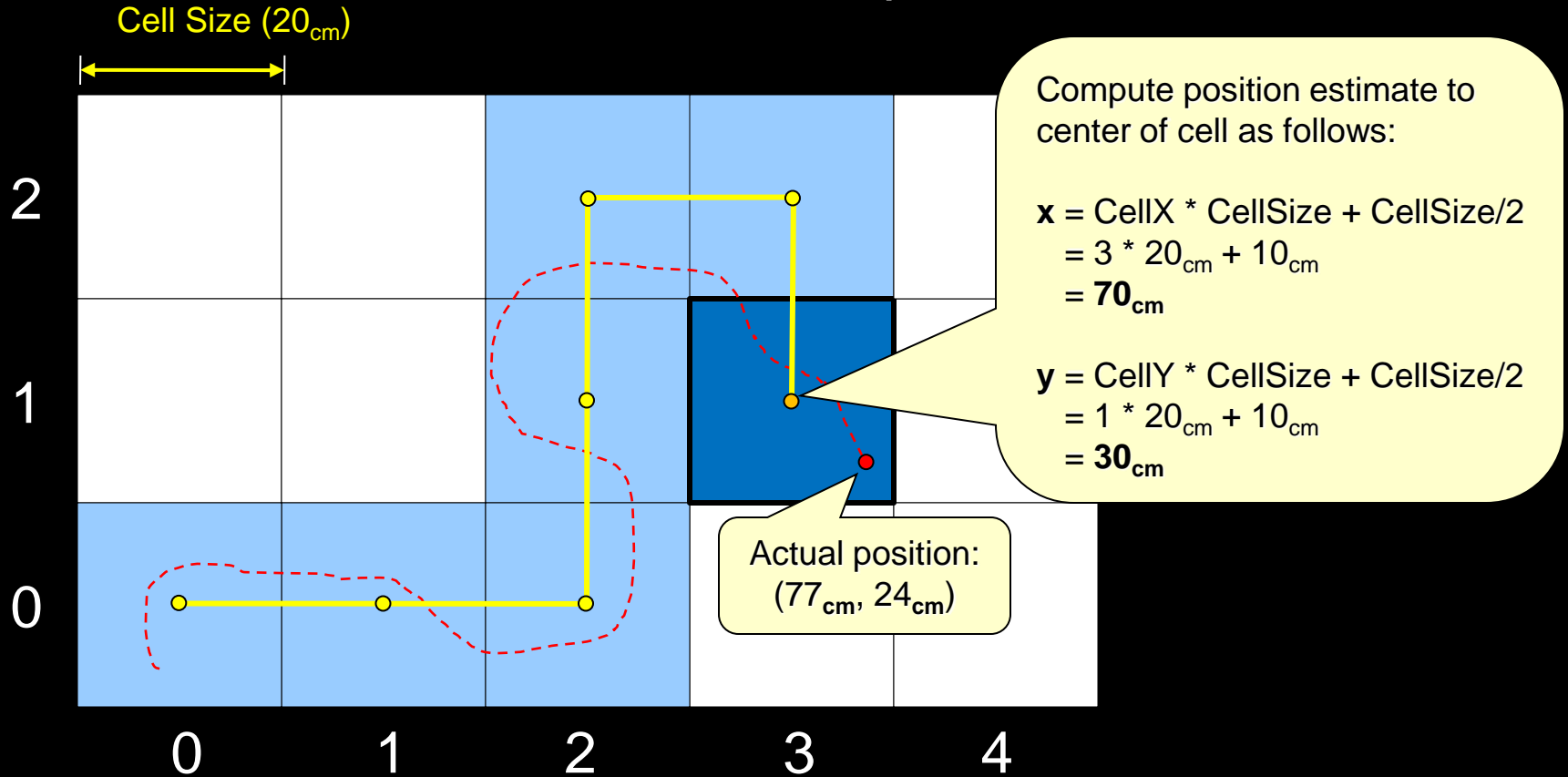
$$\begin{aligned} \mathbf{x} &= \text{CellX} * \text{CellSize} + \text{CellSize}/2 \\ &= 15 * 5_{\text{cm}} + 2.5_{\text{cm}} \\ &= \mathbf{77.5_{\text{cm}}} \end{aligned}$$

$$\begin{aligned} y &= \text{CellY} * \text{CellSize} + \text{CellSize}/2 \\ &= 4 * 5_{\text{cm}} + 2.5_{\text{cm}} \\ &= \mathbf{22.5_{\text{cm}}} \end{aligned}$$

Actual position:  
(77<sub>cm</sub>, 24<sub>cm</sub>)

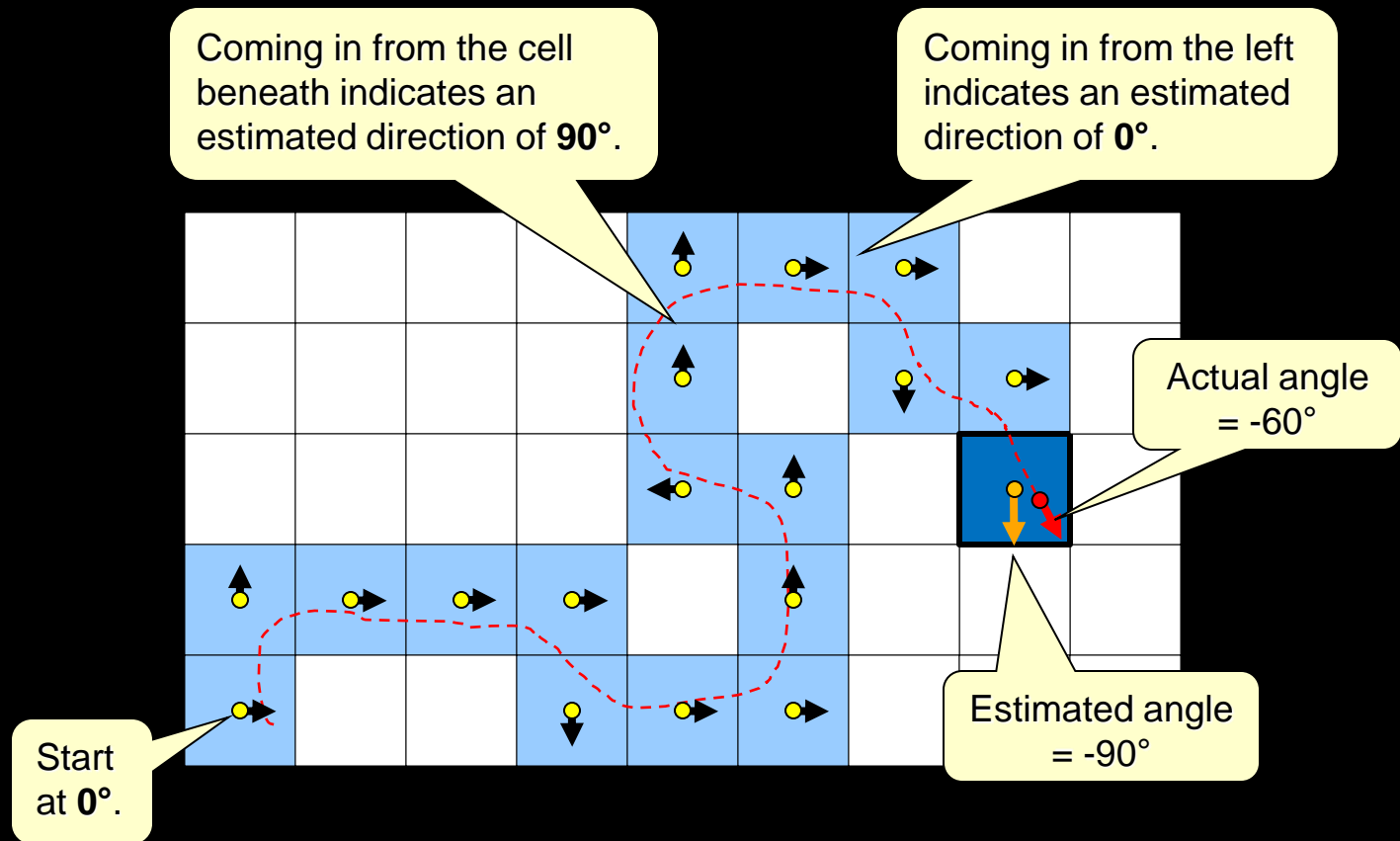
# Grid-Based Estimation

- A coarse grid will be much less accurate.
- Path will be far off from actual robot path.



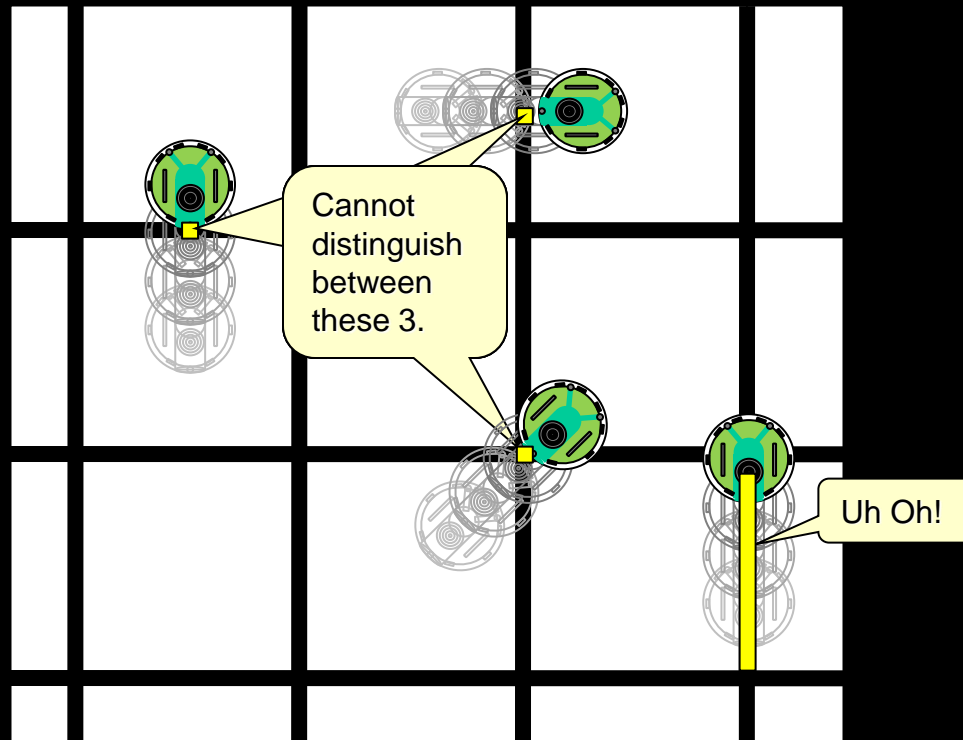
# Grid-Based Estimation

- Can also “estimate” robot’s direction when the robot moves from cell to cell:



# Detecting Grid Cell-Changes

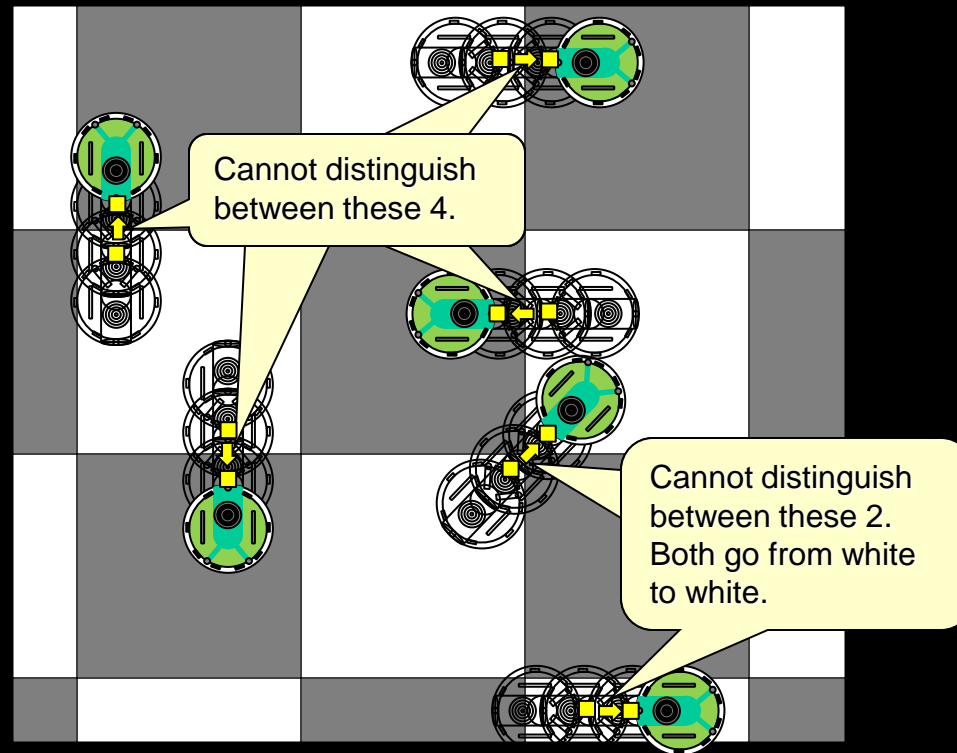
- How do we know when we cross from one cell to another ?
  - Consider crossing black lines on floor with a light sensor





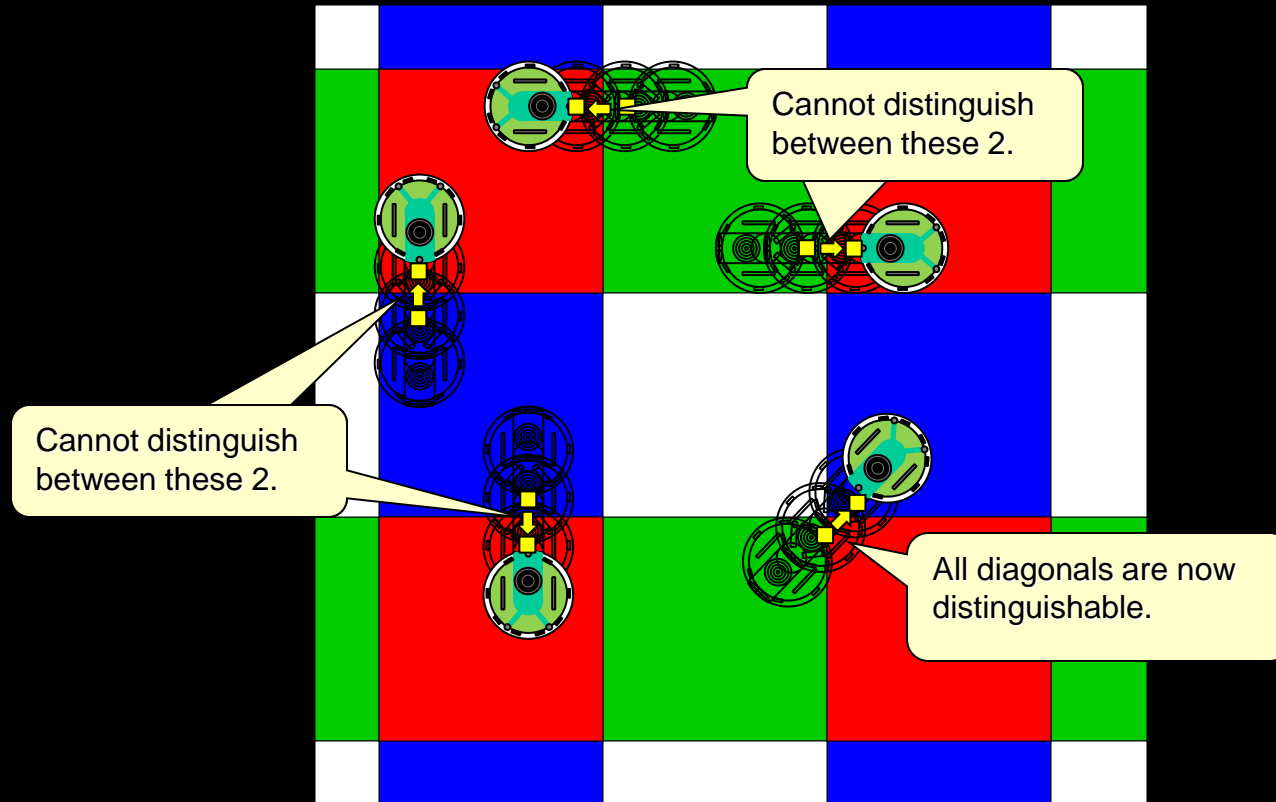
# Detecting Grid Cell-Changes

- Can color cells using checkerboard pattern:
  - Can tell when going from white to black



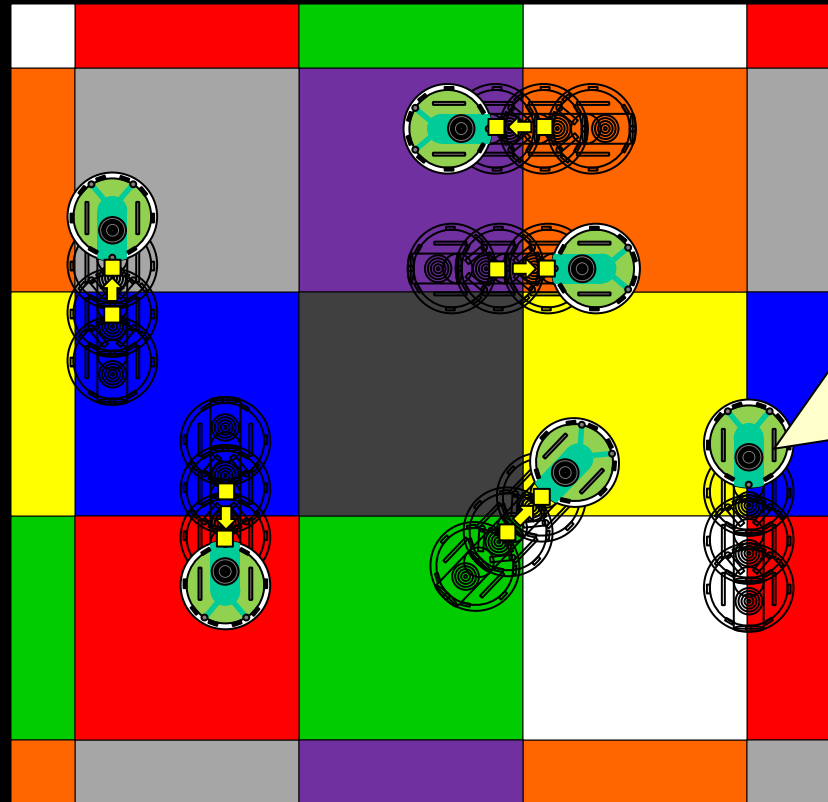
# Detecting Grid Cell-Changes

- Can use 4 colors:
  - Able to detect difference between vertical and horizontal



# Detecting Grid Cell-Changes

- Using 9 colors ensures unique cell-to-cell identification:

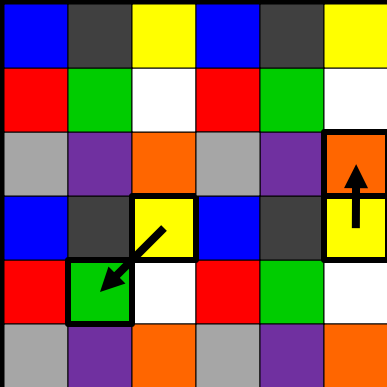


There will still be issues if the sensor lies evenly in two colors, as the reading will not match any one color.

This happens when travelling on the lines horizontally or vertically.

# Detecting Grid Cell-Changes

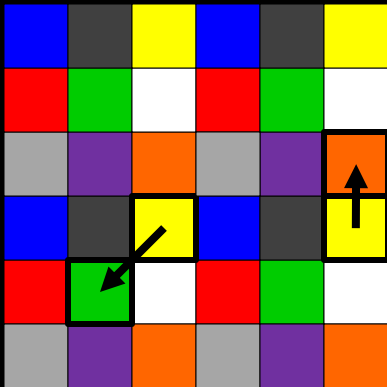
- Can define **OffsetTable** representing grid cell offsets when travelling from one color to each other color.
  - Elements in table indicate **(x,y)** cell offset when moving from cell to cell.



from\to	Red	Green	White	Blue	Black	Yellow	Gray	Purple	Orange
Red	(0,0)	(1,0)	(-1,0)	(0,1)	(1,1)	(-1,1)	(0,-1)	(1,-1)	(-1,-1)
Green	(-1,0)	(0,0)	(1,0)	(-1,1)	(0,1)	(1,1)	(-1,-1)	(0,-1)	(1,-1)
White	(1,0)	(-1,0)	(0,0)	(1,1)	(-1,1)	(0,1)	(1,-1)	(-1,-1)	(0,-1)
Blue	(0,-1)	(1,-1)	(-1,-1)	(0,0)	(1,0)	(-1,0)	(0,1)	(1,1)	(-1,1)
Black	(-1,-1)	(0,-1)	(1,-1)	(-1,0)	(0,0)	(1,0)	(-1,1)	(0,1)	(1,1)
Yellow	(1,-1)	(-1,-1)	(0,-1)	(1,0)	(-1,0)	(0,0)	(1,1)	(-1,1)	(0,1)
Gray	(0,1)	(1,1)	(-1,1)	(0,-1)	(1,-1)	(-1,-1)	(0,0)	(1,0)	(-1,0)
Purple	(-1,1)	(0,1)	(1,1)	(-1,-1)	(0,-1)	(1,-1)	(-1,0)	(0,0)	(1,0)
Orange	(1,1)	(-1,1)	(0,1)	(1,-1)	(-1,-1)	(0,-1)	(1,0)	(-1,0)	(0,0)

# Detecting Grid Cell-Changes

- Can define **AngleTable** representing estimated **angle** when travelling from one color to each other color.
  - Value in table is new angle estimate.



from\to	Red	Green	White	Blue	Black	Yellow	Gray	Purple	Orange
Red	---	0°	180°	90°	45°	135°	-90°	-45°	-135°
Green	180°	---	0°	135°	90°	45°	-135°	-90°	-45°
White	0°	180°	---	45°	135°	90°	-45°	-135°	-90°
Blue	-90°	-45°	-135°	---	0°	180°	90°	45°	135°
Black	-135°	-90°	-45°	180°	---	0°	135°	90°	45°
Yellow	-45°	-135°	-90°	0°	180°	---	45°	135°	90°
Gray	90°	45°	135°	-90°	-45°	-135°	---	0°	180°
Purple	135°	90°	45°	-135°	-90°	-45°	180°	---	0°
Orange	45°	135°	90°	-45°	-135°	-90°	0°	180°	---

# Computing Grid-Based Estimate

$x = 0_{cm}$  ,  $y = 0_{cm}$  ,  $\theta = 0^\circ$

$r$  = read ground sensor

Can be any starting location and angle.

$i_p$  = getColorIndex( $r$ ) as an int from 0-8, or 9 if no match

repeat {

move robot in some way ...

$r$  = read ground sensor

$i_c$  = getColorIndex( $r$ ) as an int from 0-8, or 9 if no match

$i_c$  = current index

if (( $i_p \neq i_c$ ) && ( $i_c \neq 9$ )) then {

Only calculate new position if color changed and new color is good

$x = x + \text{OffsetTable}[i_p][i_c].x * \text{CELL\_WIDTH}_{cm}$

$y = y + \text{OffsetTable}[i_p][i_c].y * \text{CELL\_HEIGHT}_{cm}$

cell width & height are constants

$\theta = \text{AngleTable}[i_p][i_c]$

Angle does not depend on previous angle

$i_p = i_c$

}

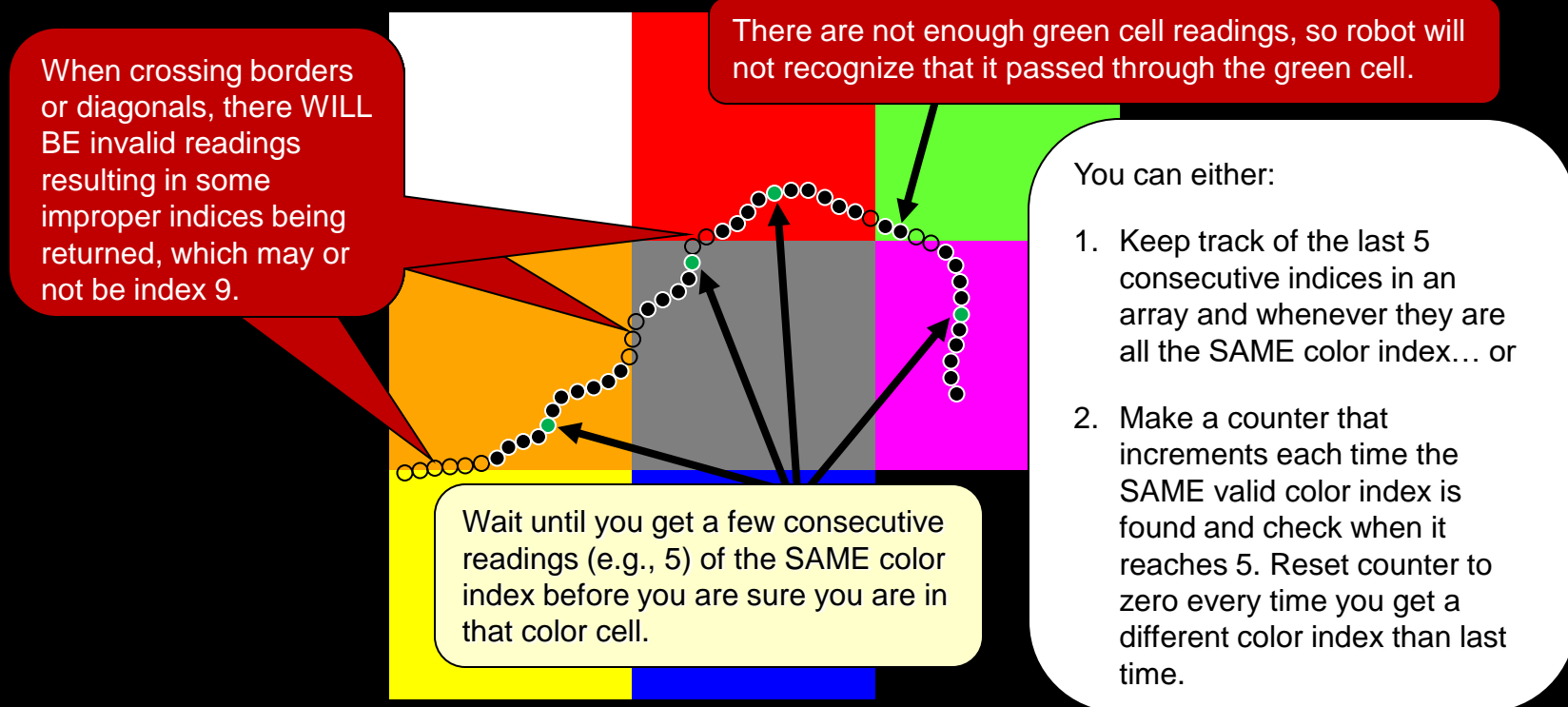
}

0	1	2
3	4	5
6	7	8

$i_p$  = previous index

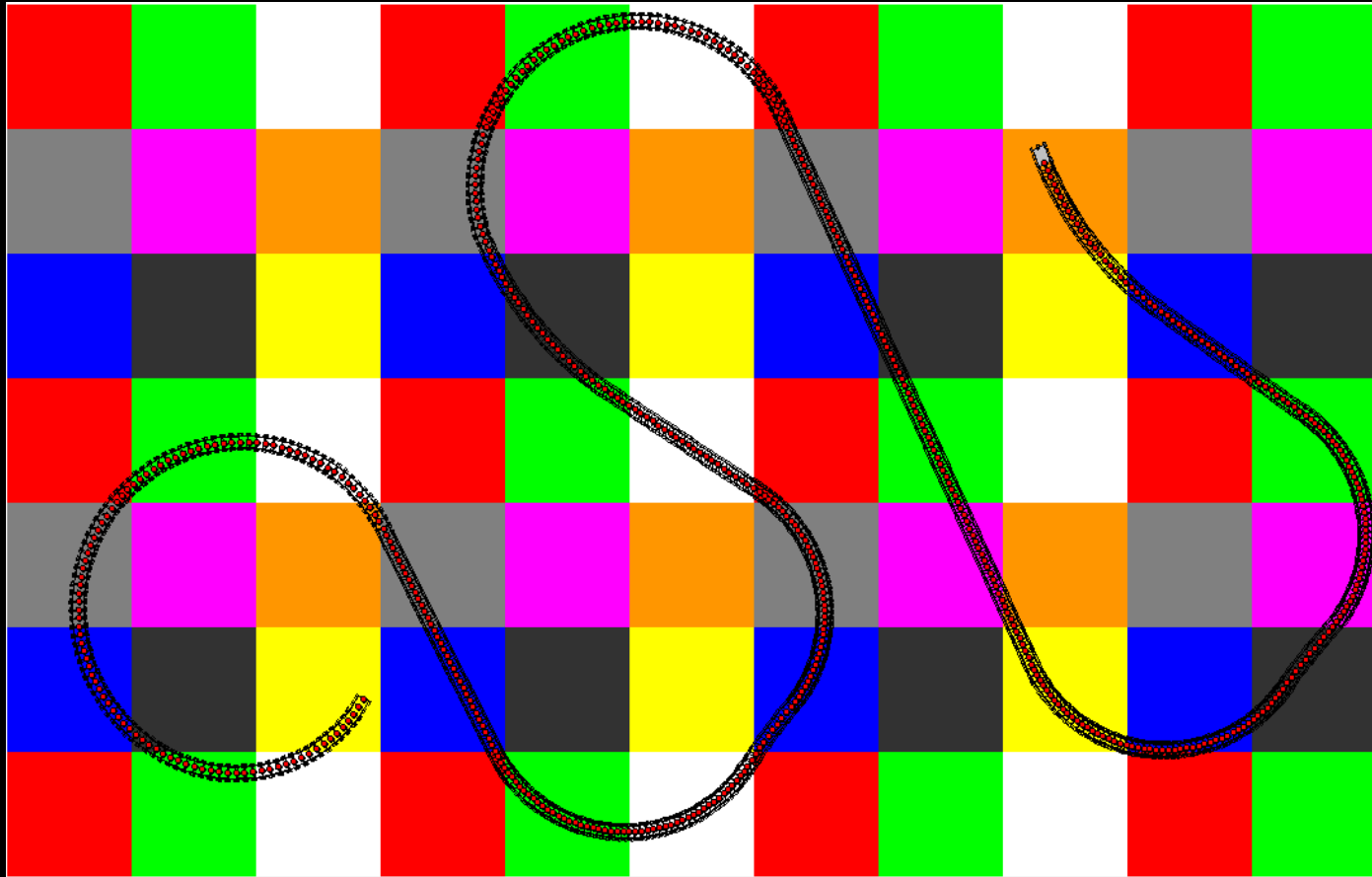
# Handling Consecutive Readings

- There will still be some problems because the robot will get a few spurious/fluctuating/invalid/wrong readings as it crosses diagonals and borders.
  - Just make sure that you have a few (e.g., 5) good readings before you are sure that you are in a cell with a certain color



# Grid-Based Experiments

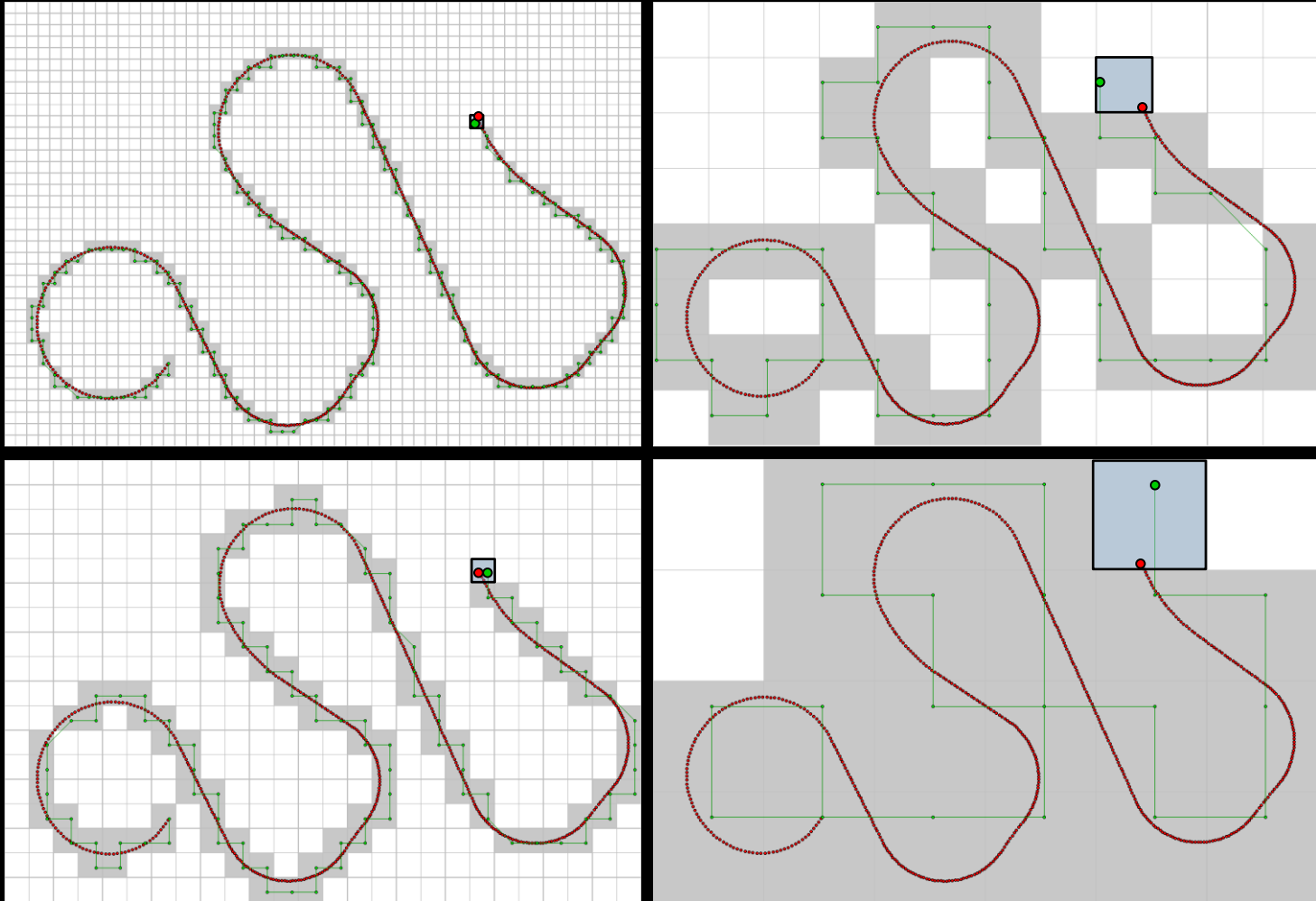
- Consider an entire floor with the colored tile pattern:





# Grid-Based Results

- Here are some results for various cell sizes:



# E-Puck Ground Sensor

- The e-puck has an expansion pack that allows 3 ground sensors to detect the amount of light under the front of the robot.

```
import com.cyberbotics.webots.controller.Device;

// Ground sensor is a Device that behaves like a DistanceSensor
DistanceSensor groundSensor;

// Go through the devices and look for sensor named "gs1"
// because it is not part of the standard e-puck robot
int numDevices = robot.getNumberOfDevices();
for (int i=0; i<numDevices; i++) {
    Device d = robot.getDeviceByIndex(i);
    if (d.getName().equals("gs1")) {
        groundSensor = (DistanceSensor)d; // Typecast is needed
        groundSensor.enable(timeStep);
    }
}

// while (...) {
//     Read the sensor like a distance sensor
//     returns value from 0 to 1000
//     double reading = groundSensor.getValue();
// }
```

You MUST read the sensor from INSIDE the WHILE loop

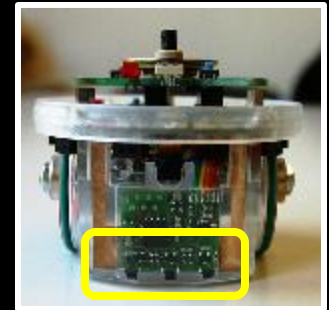
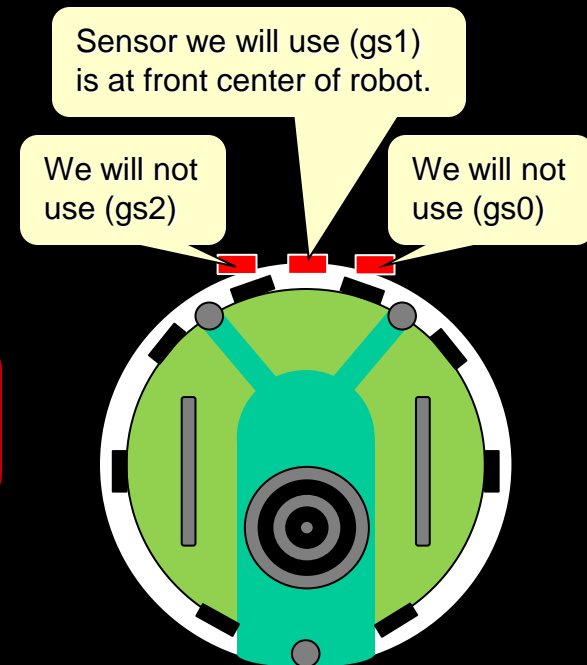


Image from  
[www.gctronic.com/doc](http://www.gctronic.com/doc)

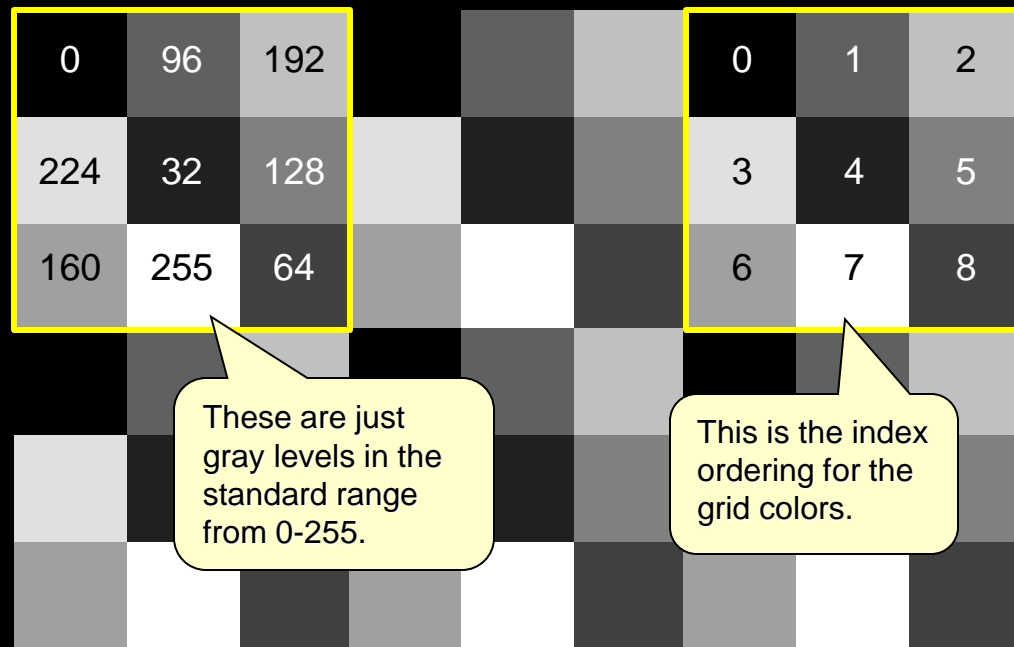


# Webots – Positioning Grid

- Cover the entire floor with a grayscale grid pattern (since ground sensors detect light intensity, not colors).
- **9 gray levels** at furthest range apart is  $255/9 \approx 32$  gray shades apart from each other.
- Shades scattered to maximize gray level changes when moving horizontal and vertical

Here is a typical ground sensor reading for the given gray shades. However, the values will fluctuate quite a bit.

300	720	822
842	525	767
800	858	644





**Start the  
Lab ...**