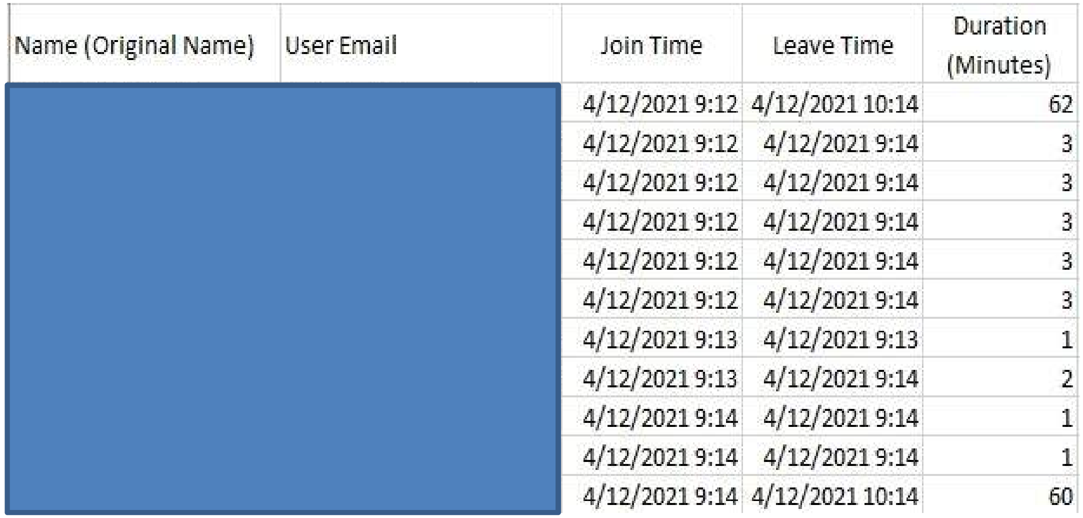
Introduction to Robotics

Felipe P. Vista IV

Class Admin Matters

Grading

- Attendance 5%



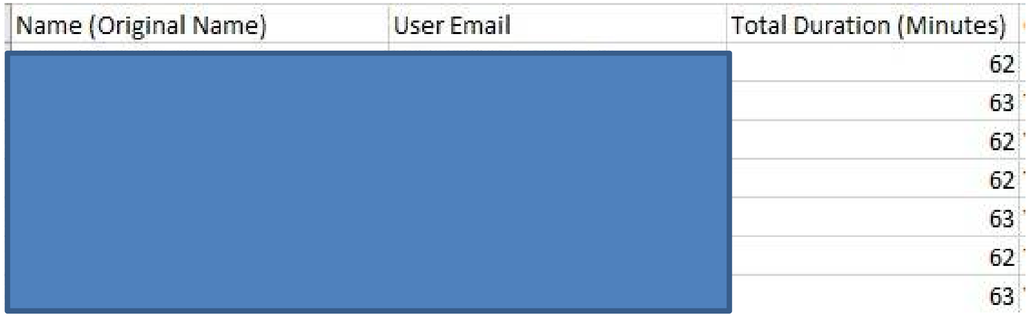
Name (Original Name)	User Email	Join Time	Leave Time	Duration (Minutes)
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 10:14	62
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 9:14	3
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 9:14	3
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 9:14	3
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 9:14	3
[redacted]	[redacted]	4/12/2021 9:12	4/12/2021 9:14	3
[redacted]	[redacted]	4/12/2021 9:13	4/12/2021 9:13	1
[redacted]	[redacted]	4/12/2021 9:13	4/12/2021 9:14	2
[redacted]	[redacted]	4/12/2021 9:14	4/12/2021 9:14	1
[redacted]	[redacted]	4/12/2021 9:14	4/12/2021 9:14	1
[redacted]	[redacted]	4/12/2021 9:14	4/12/2021 10:14	60

Bad ZOOM User Name (Absent)

- Iphone → Not your name
- SiAko 202100001 → Wrong order
- SiAko → Name only
- 202100001 → ID Num only

ZOOM User Name (Present)

- University ID Num_Name
- 202100001 SiAko → GOOD (Present)



Name (Original Name)	User Email	Total Duration (Minutes)
[redacted]	[redacted]	62
[redacted]	[redacted]	63
[redacted]	[redacted]	62
[redacted]	[redacted]	62
[redacted]	[redacted]	63
[redacted]	[redacted]	62
[redacted]	[redacted]	63

Class Admin Matters

Student Responsibilities

- Download/Install ZOOM app for online lecture
  - *Zoom profile must be your **OASIS ID+name** similar to OASIS*
  - *Ex.: **202061234 YourName***
  - *If you are asked, but no reply, then you'll be out of zoom & mark **absent***
- Regularly login, check OLD IEILMS for updates, notifications
  - *https://ieilmsold.jbnu.ac.kr*
  - *Presentations & lecture videos will be uploaded after class*
- Regularly check Kakao Group Chat for class
  - *Everybody must have a Kakao talk account*
  - *Search & add account "**botjok**", introduce yourself and name of class ("Robotics"), then you will be added to the group chat*

Intro To Robotics

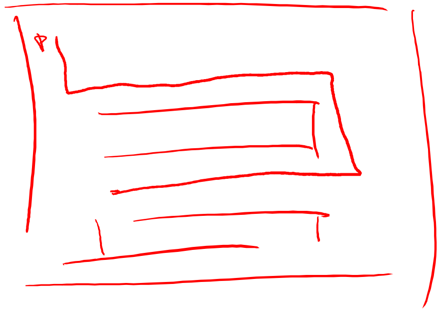
LOCAL NAVIGATION: OBSTACLE AVOIDANCE

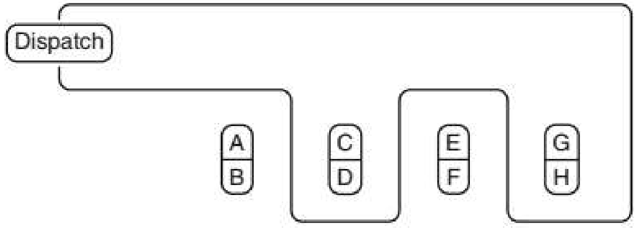
Local Navigation: Obstacle Avoidance

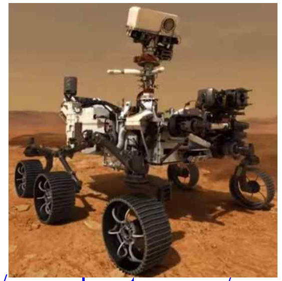
- ➢ Obstacle Avoidance
- ➢ Following a Line with a Code
- ➢ Ants Searching for a Food Source
- ➢ A Probabilistic Model of the Ants' Behaviour
- ➢ A Finite State Machine for the Path Finding Algorithm

Intro

- Mobile robot must be able to navigate
  - *Simple : follow unobstructed line in a warehouse*
  - *Difficult : unknown complex environments*
    - Rover in Mars
    - Submersible in undersea mountain range
- Ex: Self driving car
  - *Travel along a road*
  - *Other cars, obstacles on the road, pedestrian crosswalks, road construction*







https://pureadvantage.org/wp-content/uploads/2016/11/Boxfish_ROV_dive_3.jpg
https://www.slashgear.com/wp-content/uploads/2020/03/mars_2020_rover_main-1280x720.jpg

Self-driving car

- Self-driving car navigation can be divided into two tasks:

1) High-level task : find path start → goal position

   – *Old school : study maps, ask directions*

   New school : phone apps compute paths; real-time data give fastest route

   ** offline or online (GPS + real time data) → updated path*

   – *Path planning once (at start) or every few mins (w/ real-time data)*

2) Low-level task : adapt behaviour to environment

   – *Stop for pedestrian, turn at intersection, avoid obstacles*

   – *Obstacle avoidance done frequently*

   Since we never know

   - When pedestrian jump into road
   - Car ahead suddenly stops

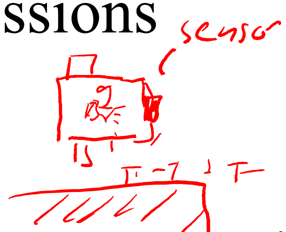
Obstacle Avoidance

- So far, we studied algorithms moving toward detected objects
  - *Obstacles blocking path likely encountered by robot moving toward goal*
- Now, assume robot can detect unobstructed path to goal
  - *Ex. Detect light on the goal*
  - *We study three obstacle (walls) avoidance algorithms*

1) Wall following
   - *Not work if multiple obstacles in environment*
2) Avoid multiple obstacles
   - *Must know general direction of goal (maybe from GPS)*
   - *Some obstacles can trap robot in a loop*
3) Pledge
   - *Improved 2nd algorithm to deal with possible loop*

Obstacle Avoidance

- The algorithms use abstract conditional expressions
  - *wall-ahead* : *wall close to front of the robot*
  - *wall-right* : *wall close to right of the robot*
  - *corner-right* : *robot moving around obstacle & sense corner to its right*

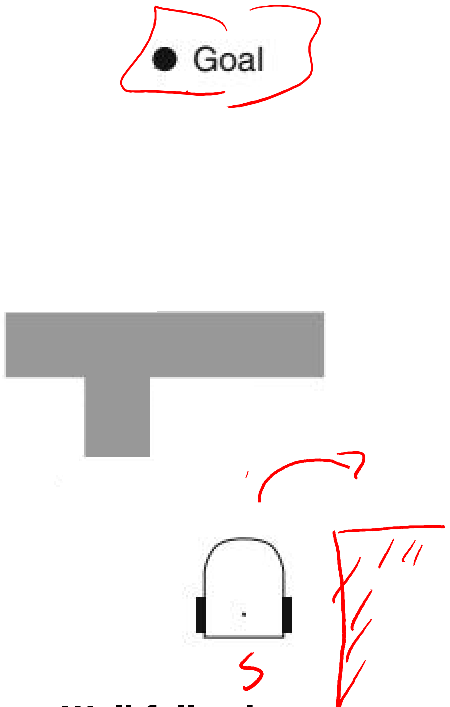


Conditional expressions implementation

- **wall-ahead** : *using horizontal proximity or touch sensor*
- **wall-right** : *sensor mounted right side of robot or rotating distance sensor. If only forward-facing sensor, robot slightly turn to right, check if wall detected, then turn back to original orientation.*
- **corner-right** : *can be as an extension of* wall-right*. When* wall-right *value switches TRUE→FALSE, make short right turn and check if* wall-right *becomes TRUE again*

Wall Following

- We have an obstacle between robot and goal
- Algorithm
  - *Maintain position so that wall always to its right –*
    wall detected ahead → turn left so wall to its right
    wall detected right → continue moving along wall
    corner detected → turn right then continue moving
  - *Continually search for goal (black dot)*



Wall following

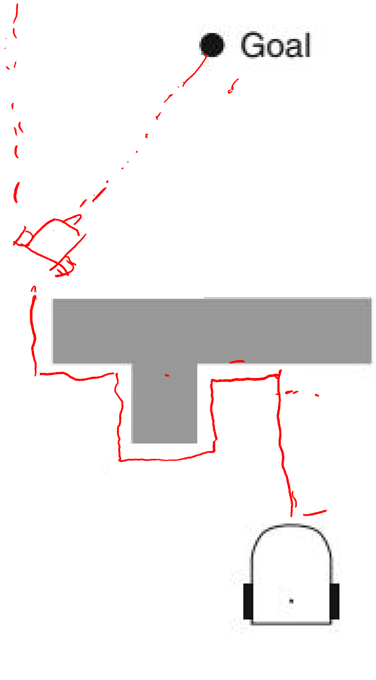
Activity 7.1: Simple Wall Following

Wall Following

- We have an obstacle between robot and goal
- Algorithm
  - *Maintain position so that wall always to its right*
    wall detected ahead → turn left so wall to its right
    wall detected right → continue moving along wall
    corner detected → turn right then continue moving
  - *Continually search for goal (black dot)*

```
while not at goal
    if goal-detected
        move towards goal
    else if wall-ahead
        turn left
    else if corner-right
        turn right
    else if wall-right
        move forward
    else
        move forward
```

Activity 7.1: Simple Wall Following



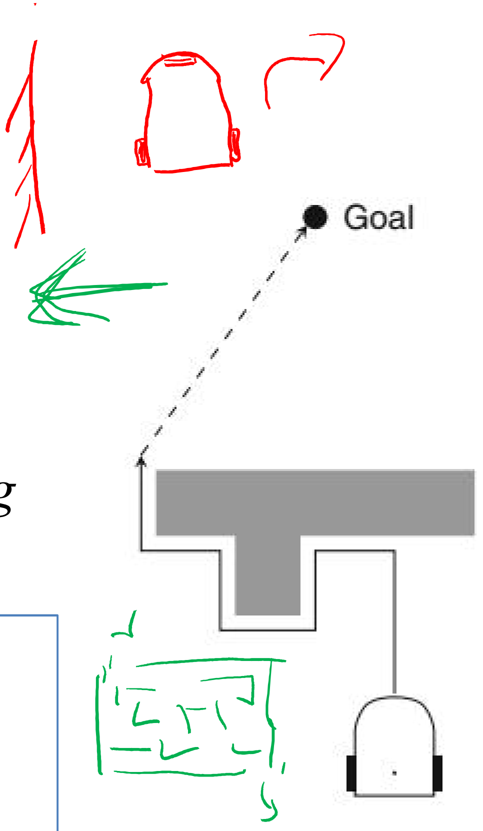
Wall following

Wall Following

- We have an obstacle between robot and goal
- Algorithm
  - *Maintain position so that wall always to its ~~right~~* left
    wall detected ahead → turn ~~left~~ right *so wall to its right*
    wall detected ~~right~~ left *→ continue moving along wall*
    left- *corner detected → turn ~~right~~* left *then continue moving*
  - *Continually search for goal (black dot)*

```
while not at goal
    if goal-detected
        move towards goal
    else if wall-ahead
        turn ~~left~~ right
    else if corner-~~right~~ left
        turn ~~right~~ left
    else if wall-~~right~~ left
        move forward
    else
        move forward
```

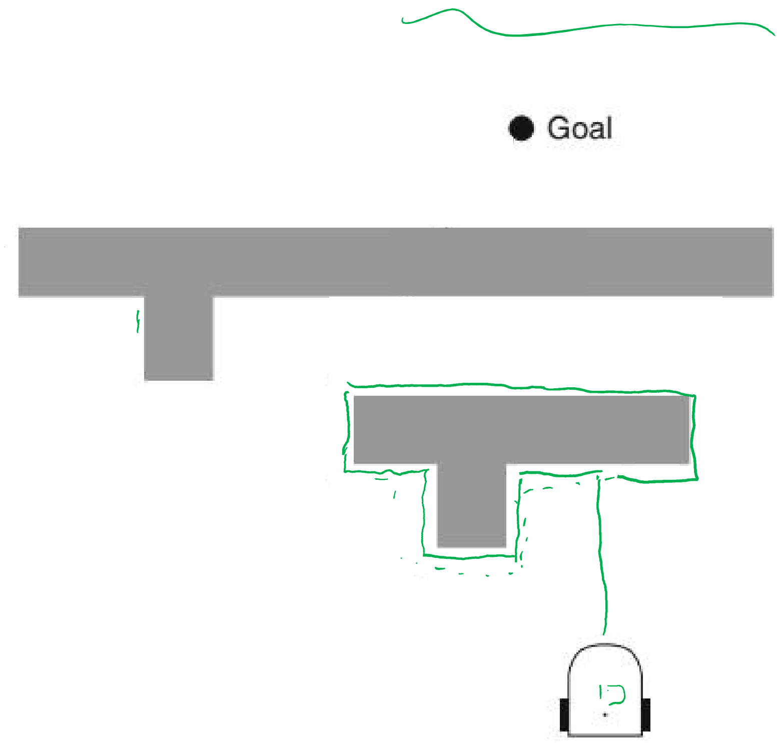
Activity 7.1: Simple Wall Following



Wall following

Wall Following

- We have two obstacles between robot and the goal



Using Simple Wall-Following

- Maintain position so wall always to its right
  - wall detected ahead

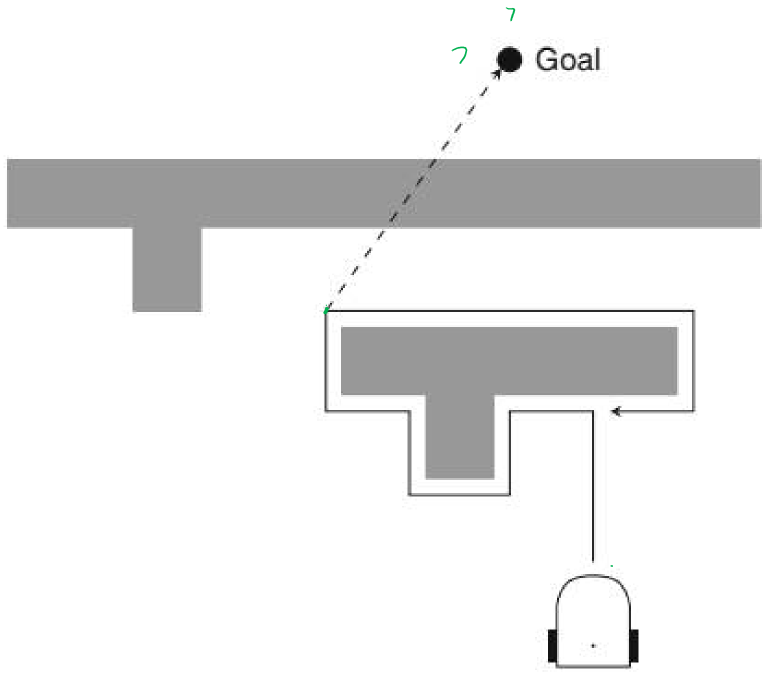
    → turn left so wall to its right
  - wall detected right

    → continue moving along wall
  - corner detected

    → turn right then continue moving
- Continually search for goal (black dot)

Wall Following

- We have two obstacles between robot and the goal
  - *Robot cannot detect goal, it will move around 1st obstacle indefinitely*



Using Simple Wall-Following

- Maintain position so wall always to its right
  - wall detected ahead
    → turn left so wall to its right
  - wall detected right
    → continue moving along wall
  - corner detected
    → turn right then continue moving
- Continually search for goal (black dot)

Simple wall following doesn't always enable robot to reach the goal ☹...

Wall Following with Direction

- Simple Wall-Following algorithm
  - *A local algorithm that only looks at its immediate environment*
  - *Don't consider that higher-level navigation algorithm knows roughly direction robot must take to reach goal*
- Modified algorithm
  - *Similar to wall following algorithm*
  - *Except preference to move in heading to goal if possible*
  - *no obstacle → move at heading going to goal*

    robot can't move in heading to goal → use wall following

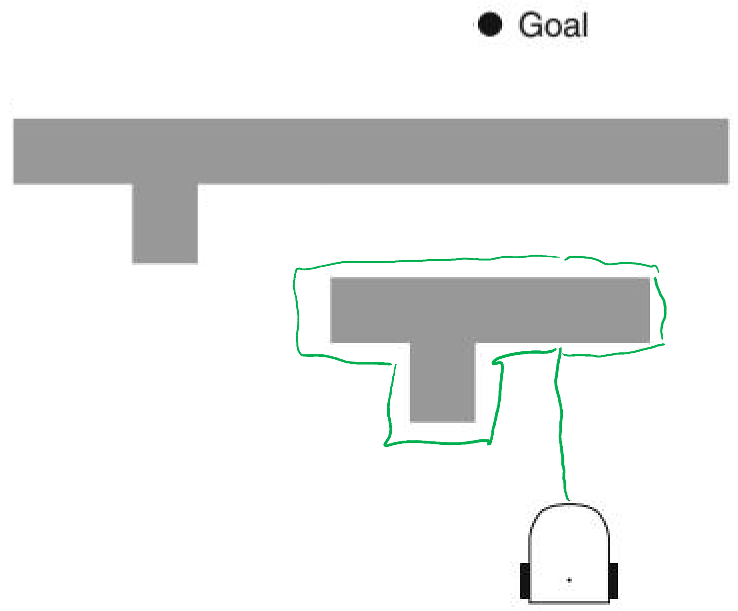
    → *variable* heading *to store current heading*

    heading == heading to goal → move forward instead of looking for corner

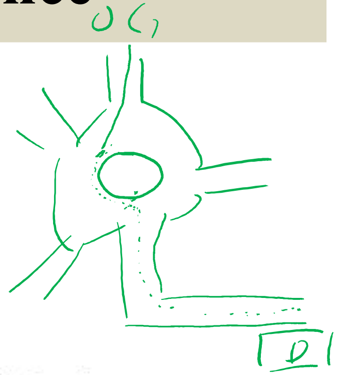
    current

Wall Following with Direction

- We have two obstacles between robot and the goal



Using Wall-Following w/ Direction

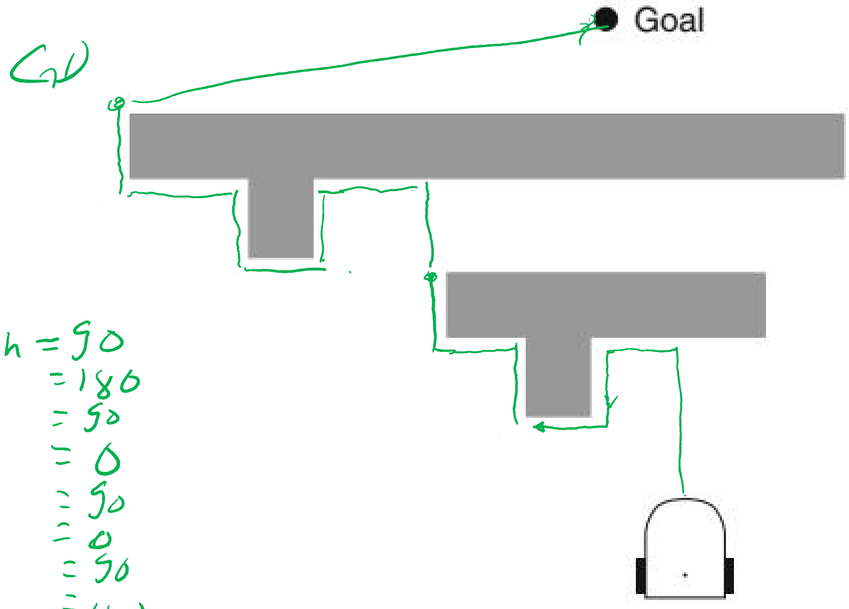


Wall Following with Direction

- We have two obstacles between robot and the goal

```
integer heading ← 0° // relative to north

while not at goal
  if goal-detected
    move towards goal
  else if wall-ahead
    turn left
    heading ← heading + 90°
  else if corner-right
    if heading = multiple of 360°
      move forward
    else
      turn right
      heading ← heading - 90°
  else if wall-right
    move forward
  else
    move forward
```



h = 90
= 180
= 90
= 0
= 90
= 0
= 90
= 180
= 90
= 0
= 90

= 0

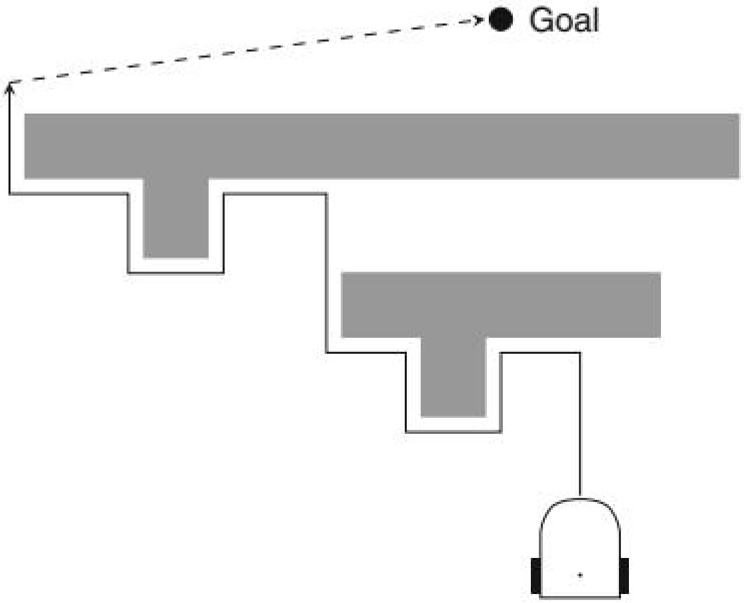
Using Wall-Following w/ Direction

Wall Following with Direction

- We have two obstacles between robot and the goal

```
integer heading ← 0° // relative to north

while not at goal
   if goal-detected
      move towards goal
   else if wall-ahead
      turn left
      heading ← heading + 90°
   else if corner-right
      if heading = multiple of 360°
         move forward
      else
         turn right
         heading ← heading - 90°
   else if wall-right
      move forward
   else
      move forward
```



Using Wall-Following w/ Direction

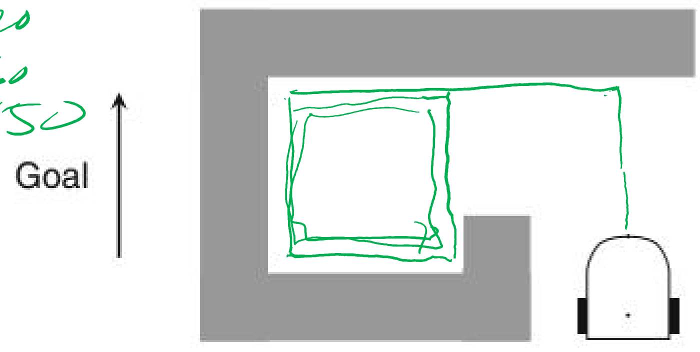
Wall Following with Direction G

- We have a G-shaped obstacle between robot and goal

```
integer heading ← 0° // relative to north

while not at goal
   if goal-detected
      move towards goal
   else if wall-ahead
      turn left
      heading ← heading + 90°
   else if corner-right
      if heading = multiple of 360°
         move forward
      else
         turn right
         heading ← heading - 90°
   else if wall-right
      move forward
   else
      move forward
```

h = 90
= 180
= 220
= 360
= 450



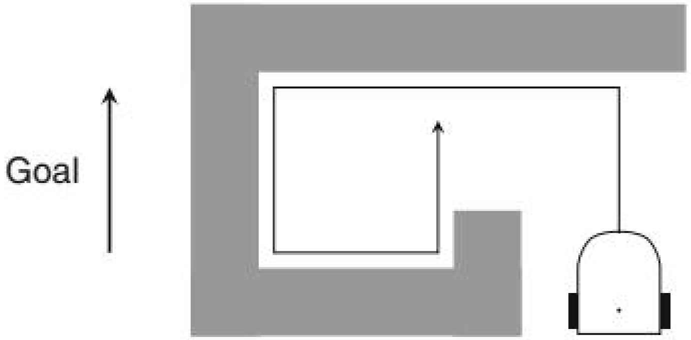
Using Wall-Following w/ Direction

Wall Following with Direction

- We have a G-shaped obstacle between robot and goal
  - *heading = 360 after 4 turns → continue move forward → follow wall again*

```
integer heading ← 0° // relative to north

while not at goal
   if goal-detected
      move towards goal
   else if wall-ahead
      turn left
      heading ← heading + 90°
   else if corner-right
      if heading = multiple of 360°
         move forward
      else
         turn right
         heading ← heading - 90°
   else if wall-right
      move forward
   else
      move forward
```



Using Wall-Following w/ Direction

Wall following w/ Direction fails if it encounters a G-shaped obstacle ☹...

Pledge

- Modify LINE 8 of Wall Following w/ Direction algorithm

```
08:      if heading = multiple of 360°   →     if heading = 0°
```

- Modified algorithm
  - ***IFF*** *(cumulative_heading == 0° &&* ***NOT*** *moving NORTH (heading multiple of 360°)) → move forward*
  - *"G-shaped" obstacle now avoided*
  - *When corner encountered (black dot), it is moving north, but its heading is 360° after four left turns*
  - *360° multiple of 360° <> 0°*
  - *Therefore, continue following wall until four right turns (will subtract 360°) → total heading is 0°.*

Local Navigation: Obstacle Avoidance

Pledge

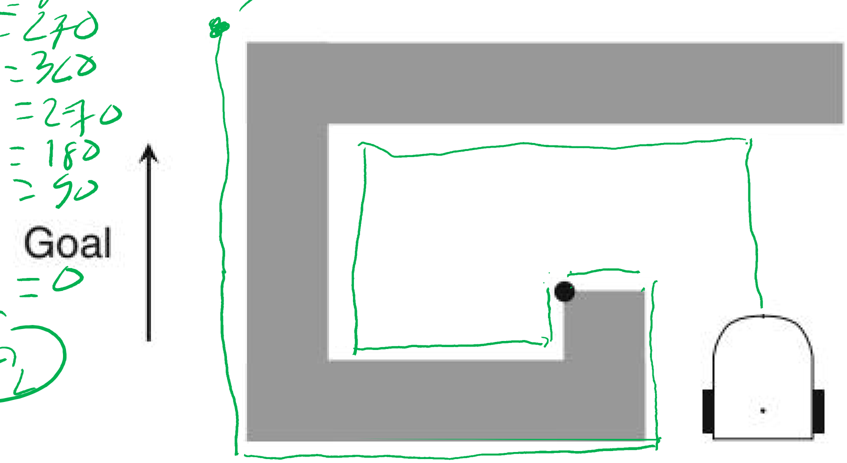
- We have a G-shaped obstacle between robot and goal

270

```
integer heading ← 0° // relative to north

while not at goal
  if goal-detected
    move towards goal
  else if wall-ahead
    turn left
    heading ← heading + 90°
  else if corner-right
    if heading = 0°    // h = multiple of 360
      move forward
    else
      turn right
      heading ← heading - 90°
  else if wall-right
    move forward
  else
    move forward
```

h=90
=180
=270
=360
=270
=180
=90
=0



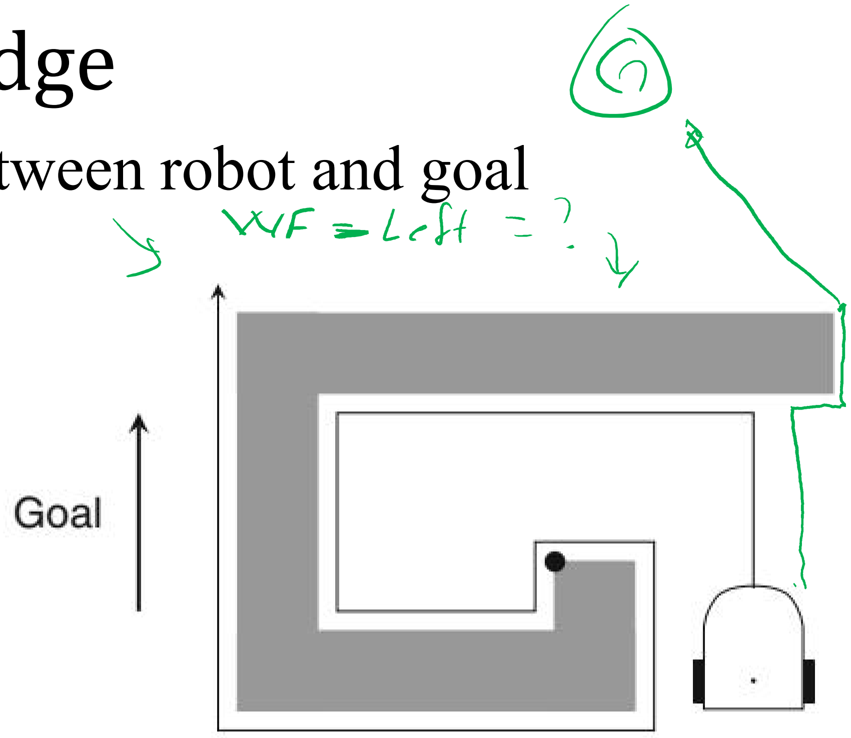
Using Pledge for Wall Following

?????

Pledge

- We have a G-shaped obstacle between robot and goal

```
integer heading ← 0° // relative to north
while not at goal
    if goal-detected
        move towards goal
    else if wall-ahead
        turn left
        heading ← heading + 90°
    else if corner-right
        if heading = 0°
            move forward
        else
            turn right
            heading ← heading - 90°
    else if wall-right
        move forward
    else
        move forward
```



WF = Left = ?

Using Pledge for Wall Following

Now, we are able to navigate to the GOAL!!!

Local Navigation: Obstacle Avoidance

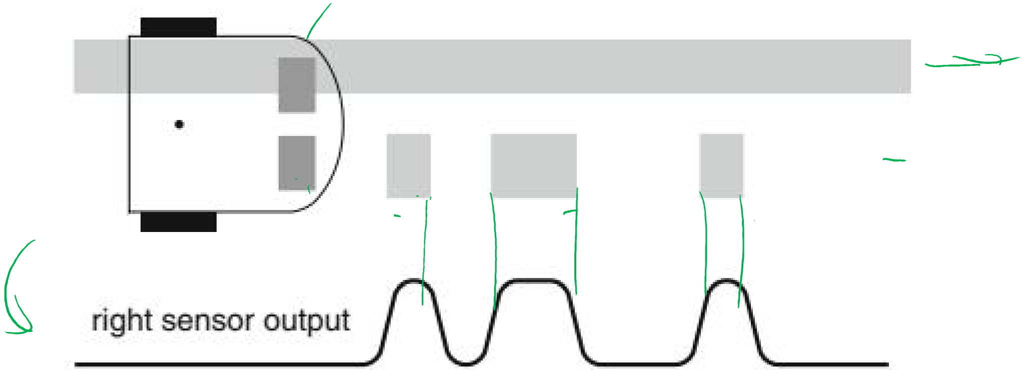
- Obstacle Avoidance
- Following a Line with a Code
- Ants Searching for a Food Source
- A Probabilistic Model of the Ants' Behaviour
- A Finite State Machine for the Path Finding Algorithm

Following a Line with a Code

- Line following algorithm
  - *Guide robot within an environment*
  - *But it is not navigation*
- To navigate
  - *Need localization algorithm to know when we reached goal*
  - *But do not need continuous localization algorithm (next lecture)… only need to know positions on the line that facilitate fulfilling task*
- Similar to navigating while driving
  - *Need to know : interchanges, intersections, major landmarks, so on… → to know where we are*
  - *Between such positions → just follow the road*

Following a Line with a Code

- Navigation w/o continuous localization
  - *Possible by reading code placed on the floor beside the line*

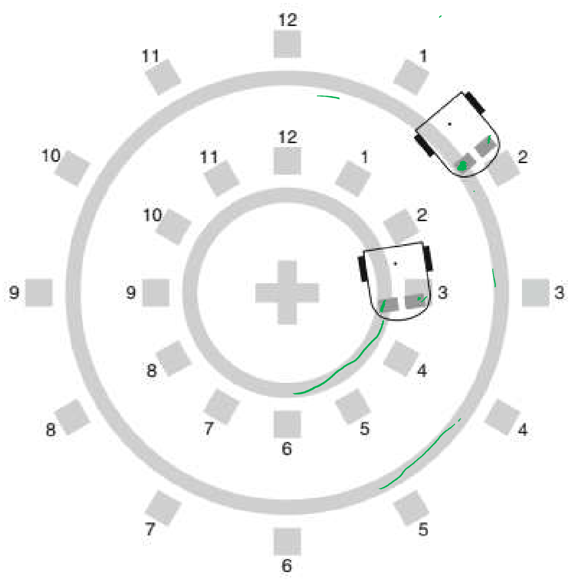


Robot following line +code

- Robot with two ground sensors
  - *left sensor → follow the line*
  - *right sensor → read the code*
- Signals from both sensors shown below robot & line+code

Following a Line with a Code

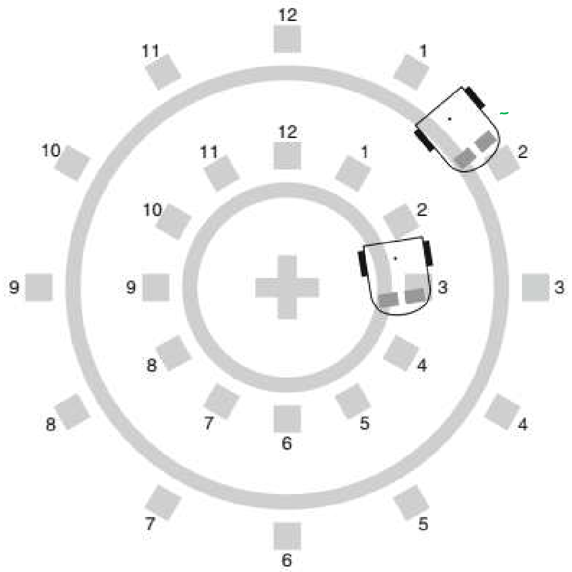
- Circular line following while reading a code
- Ex. 1 : Robotic Clock
  - Robot #1 → indicate ~~hour~~ minute
  - Robot #2 → indicate ~~minute~~ hour
- Ex. 2 : Alternate Implementation
  - Robot #1 → Complete rev in 1 hour
  - Robot #2 → Complete rec in 1 day
- Any difference between the two implementations?
  - ????????



Robotic line+code following

Following a Line with a Code

- Circular line following while reading a code
- Ex. 1 : Robotic Clock
  - *Robot #1 → indicate ~~hour~~* minute
  - *Robot #2 → indicate ~~minute~~* hour
- Ex. 2 : Alternate Implementation
  - *Robot #1 → Complete rev in 1 hour*
  - *Robot #2 → Complete rec in 1 day*
- Any difference between the two implementations?
  - ~~Robot movement in Ex. 1 dependent on each other while in Ex.2 are not…~~
    1 rev of Ex1.R1 → 1/60th step increment for Ex1.R2
    Ex2.R1 faster than Ex2.R2

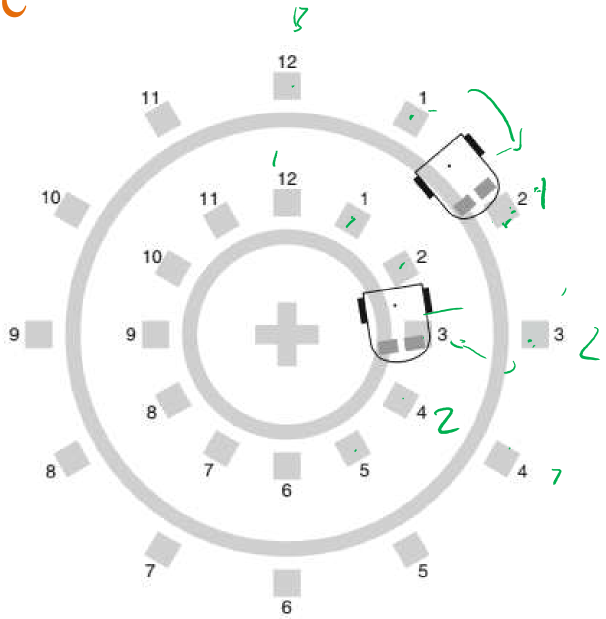


Robotic line+code following

1 ~ 12

Following a Line with a Code

- Circular line following while reading a code
- **Ex. 1** : Robotic Clock
  - Robot #1 → *indicate hour*
  - Robot #2 → *indicate minute*
- **Ex. 2** : Alternate Implementation
  - Robot #1 → *Complete rev in 1 hour*
  - Robot #2 → *Complete rec in 1 day*



Robotic line+code following

Any way to improve given configuration?

- *For Ex.1 : Possible to get time few minutes after starting from given current state?*
- *Same with Ex.2*

Local Navigation: Obstacle Avoidance

- ➢ Obstacle Avoidance
- ➢ Following a Line with a Code
- ➢ Ants Searching for a Food Source
- ➢ A Probabilistic Model of the Ants' Behaviour
- ➢ A Finite State Machine for the Path Finding Algorithm

Ants Searching for a Food Source

- If line exists + localization (like code) then previous algorithm can be used
- High level algorithm of finding a path
  - *No line? → robot create its own line!*
  - *Don't need to know its location in environment (GPS)*
  - *Use landmarks in environment to navigate*
- We study this in context of ants searching for food

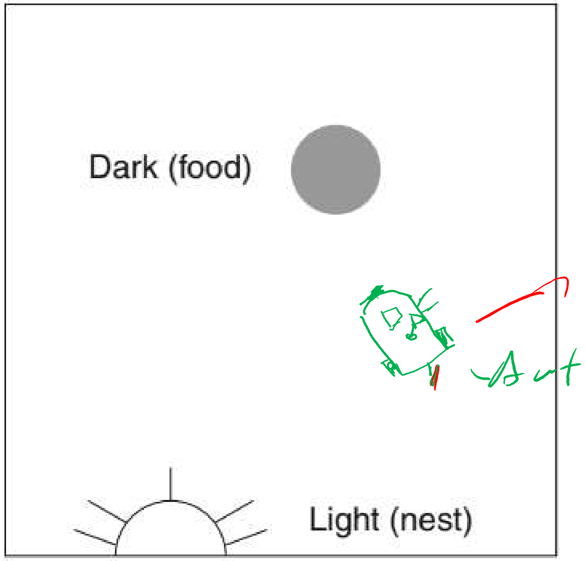
Ants Searching for a Food Source

(General Description)

- There's a nest of ants. The ants search randomly for source of food.
- When an ant finds food → it returns directly to the nest by using landmarks & its memory of the path it took from nest to source of food.
- Ant going back to nest from source of food deposit chemical pheronomes along the way
- More ants find food source then return to nest → trail accumulates more pheronomes than other areas ants visit
- Eventually, amount of pheronomes in trail is so strong → ants can follow direct path from nest to source of food

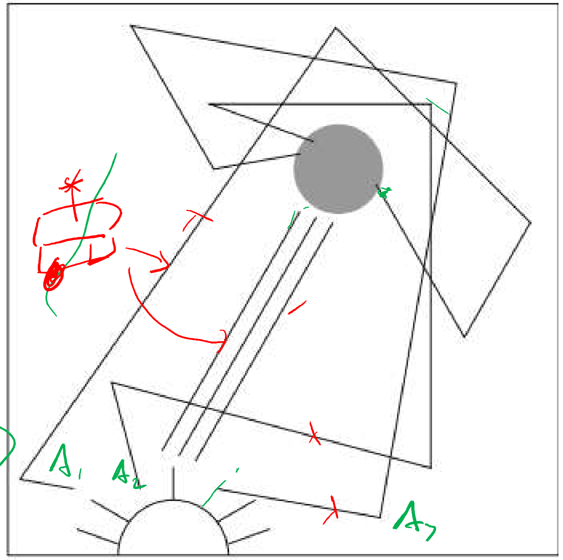
Ants Searching for a Food Source

- (b) three random trails → discover food source → ants return directly to nest → leave three straight lines (pheronomes)
  - ***Concentrated*** *pheronomes used to find food source directly*



Ant

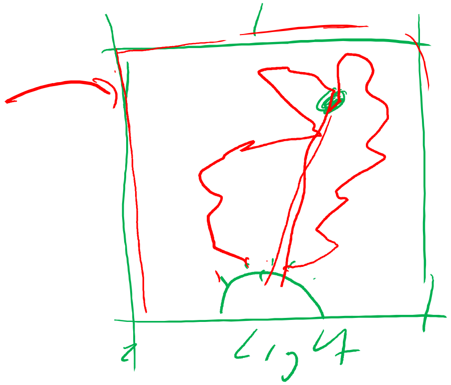
(a) Ants nest & food source



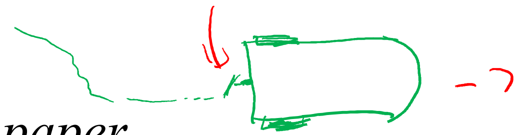
(b) Phereonomes creating a trail

Locating the Nest

- Robot setup → ANT
  - *Assume fixed area where robot can move*
  - *Ground sensor : detect food source = dark spot*
  - *Proximity sensor : detect walls of the area*



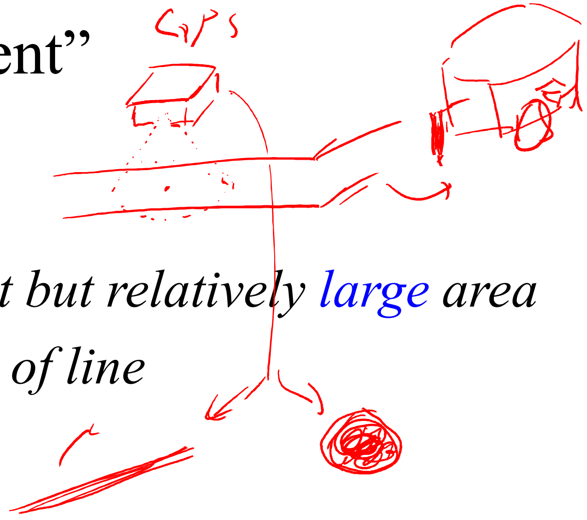
- Nest setup
  - *Accelerometer : area is a slope, nest at lowest point of area*
  - *Light sensor : nest is a light source* Robot (Ant)
    detected by light sensor regardless of position & heading of robot



- Pheronomes simulation
  - *Cover area w/ white sheet of paper*
  - *Attach black marker to robot → to draw line wherever it moves*
  - *Ground sensor : detect marks in the area*

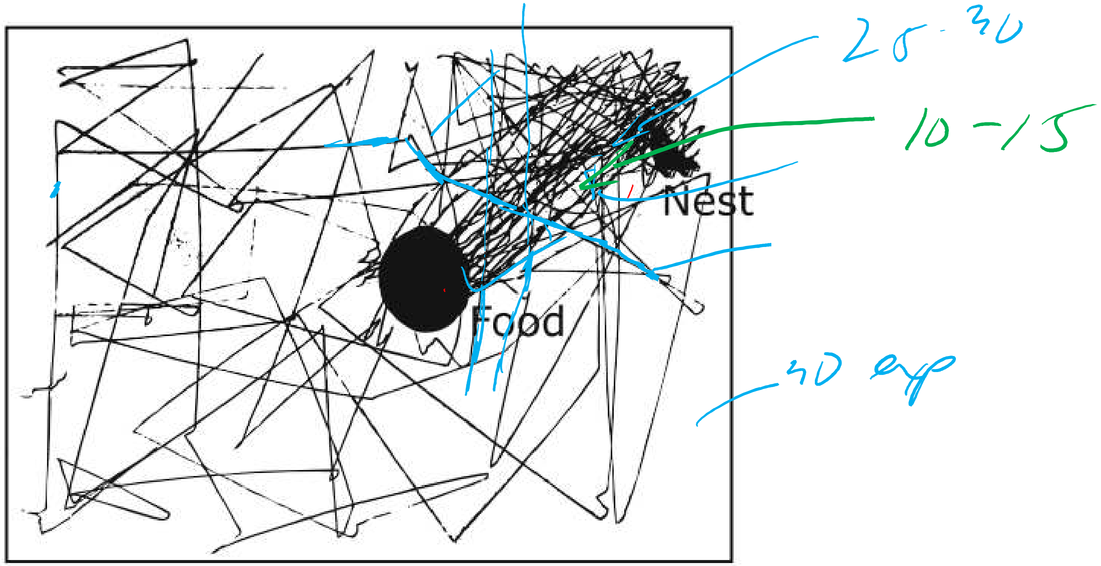
Sensing Areas of High Density

- Recall "Line Following Without a Gradient"
  - *Ground proximity sensor has aperture*
    - "opening" through w/c light enters
  - *Sensors don't sense single geographical point but relatively large area*
  - *Perform experiments to obtain optimal width of line*



- Optimal width of marker
  - *Too thin → trail not detected;*
  - *Too thick → random movement markings might be mistaken as part of trail*
- Food source
  - *Relatively large totally black spot →*
    gives a minimal reading of the ground sensor
    - Can be read at lowest possible value (RANGE)

Sensing Areas of High Density



- Trail between nest & food source has high density
- Effective threshold bet. trail & areas of random motion outside trail
  - *Define through experimenting with various number of lines*
- Try to make darker lines
  - *By varying its motion or by moving back & forth along the trail*

Local Navigation: Obstacle Avoidance

- Obstacle Avoidance
- Following a Line with a Code
- Ants Searching for a Food Source
- A Probabilistic Model of the Ants' Behaviour
- A Finite State Machine for the Path Finding Algorithm

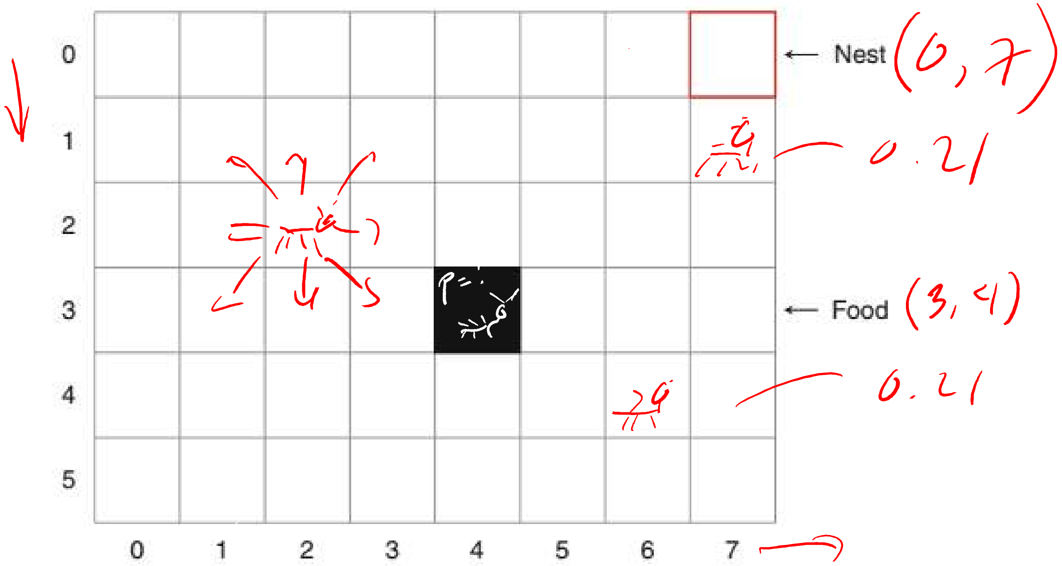
Probabilistic Model of the Ant’s Behaviour

- Model
  - *Abstraction of a system*
  - *Shows how parameters impact phenomena*
- **Ex.:**

  vol, dir, vel, # tl, timing
  - *study traffic patterns* → *predict effect of new roads or traffic lights*
- Fundamental characteristic of ant’s behavior
  - *They don’t have a map of their environment*
  - *Must move randomly* → *in order to search for food source*
  - *Therefore, their model of behaviour must be probabilistic*

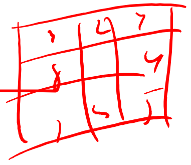
Probabilistic Model of the Ant's Behaviour

- Assume environment is rectangular area that is grid of cells (6x8)
  - *Coordinates are in (row, column)*
  - *Rows → numbered from top to bottom (like matrices in math)*
  - *Column → numbered left to right*
  - *Numbering starts from "0" (like array data in computer science)*



Probabilistic Model of the Ant’s Behaviour

- No information how ants choose their movements
  - *Assume move in any direction with same probability*

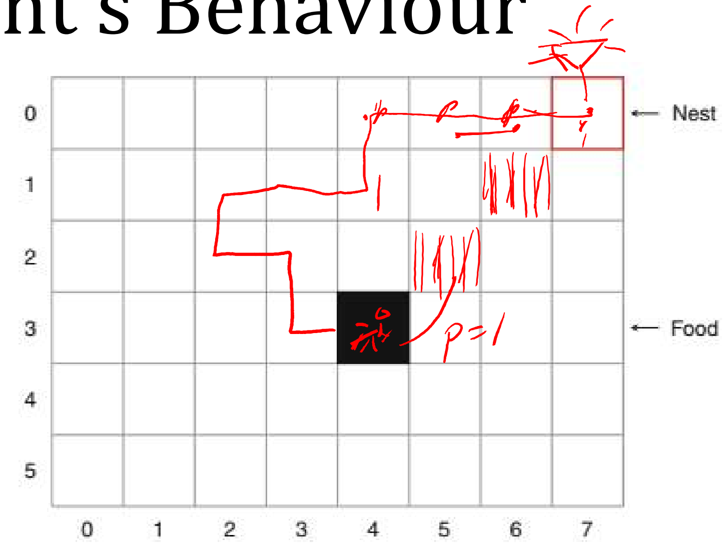


- Probability (***p***) of ant being in any cell
  - *1 divided by total number of cells*

$$p = \frac{1}{\text{total num of cells}} = \frac{1}{48} = 0.021$$

- Probability that ant is in cell with food source is ***p***
  - *Same as for any other cell*
- According to specification of ant’s behavior
  - *Ant enters cell → identify this cell with food source → return directly to nest*

Probabilistic Model of the Ant’s Behaviour



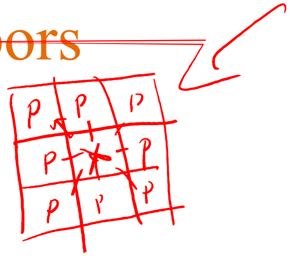
- Food source:(3,4) → Nest:(0,7)
  - *Must pass through (2,5) & (1,6)*
  - *Therefore:* (3,4) → (2,5) → (1,6) → (0,7)
- Probability ant is in any of three cells?
  - Two possibilities:
  1) *Randomly moved there w/ probability **p***
  2) *Moved to food source randomly w/ probability **p** & then w/ probability **1** moved towards the nest*
- Total probability being in any of these cells

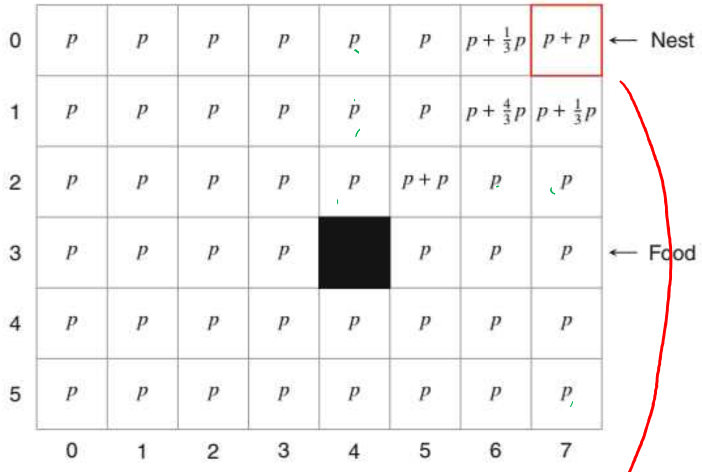
$$p + p \times 1 = p + p = 2p$$

- If lines drawn while moving → diagonal cells 2x darker than other

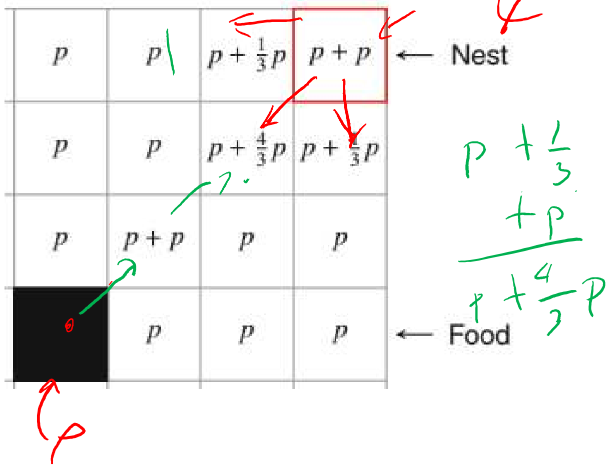
Probabilistic Model of the Ant's Behaviour

- Food source → nest → move to neighbor
  - *Select random neighbor to move to*
- Generally, cell has 8 neighbors
  - *above & below*
  - *left & right*
  - *four diagonals*
  - *Probability **p/8** in any of the neighbours*
- Nest is in the corner
  - *Only three neighbours → probability **p/3***





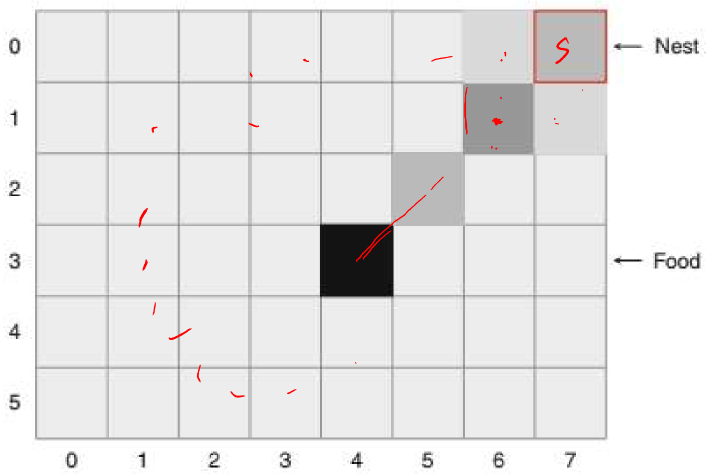
0	p	p	p	p	p	p	$p+\frac{1}{3}p$	$p+p$	← Nest
1	p	p	p	p	p	p	$p+\frac{4}{3}p$	$p+\frac{1}{3}p$	
2	p	p	p	p	p	$p+p$	p	p	
3	p	p	p	p	■	p	p	p	← Food
4	p	p	p	p	p	p	p	p	
5	p	p	p	p	p	p	p	p	
	0	1	2	3	4	5	6	7	



p	p	$p+\frac{1}{3}p$	$p+p$	← Nest
p	p	$p+\frac{4}{3}p$	$p+\frac{1}{3}p$	
p	$p+p$	p	p	
■	p	p	p	← Food

$$\frac{p + \frac{1}{3} + p}{p + \frac{4}{3}p}$$

What can we Conclude from this Model?



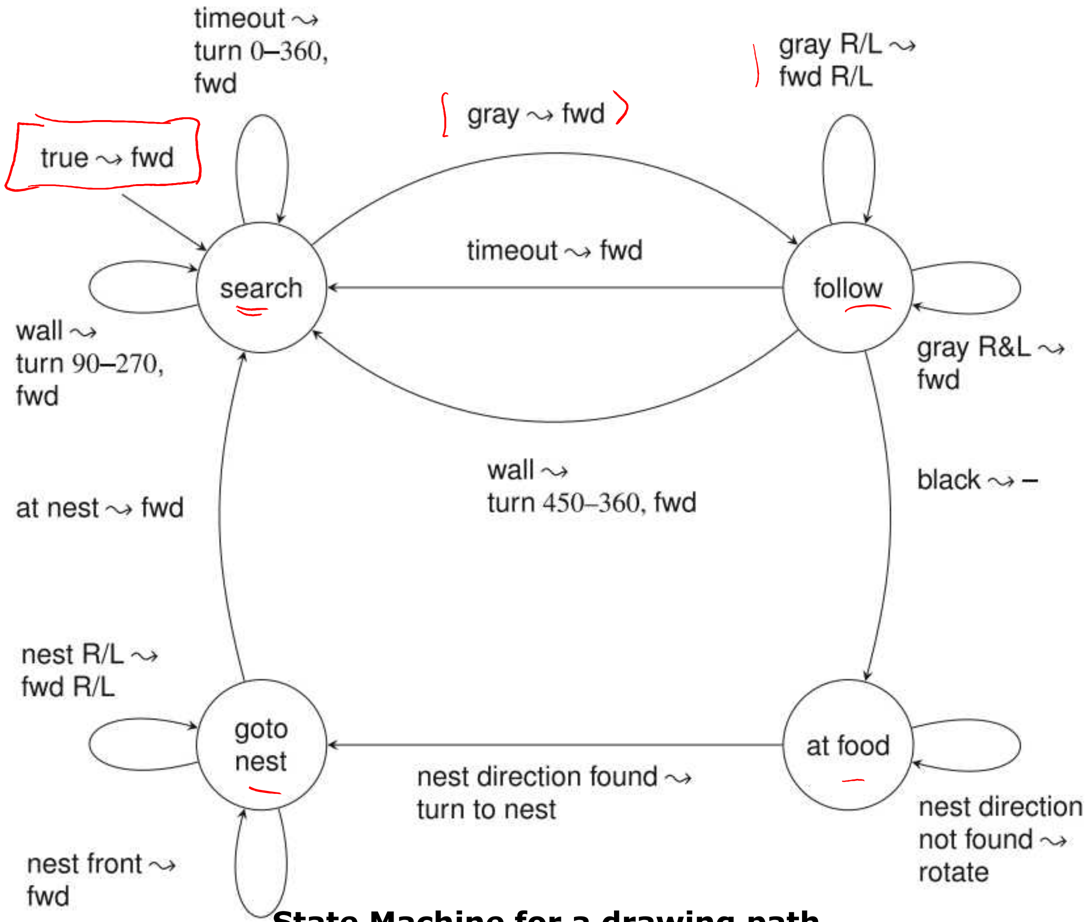
t =(x,y) ?

- Robot with marker implementation
  - *Cells w/ higher probability → darker*
- Probability of being on diagonal higher
  - *Than anywhere else in the environment*
  - *Because of behavior of finding food source then go back to nest, even if move randomly*
- Since pheronomes (black marks) at visited cell
  - *Diagonal path (food source to nest) → darker than marks on other cells*
  - *Eventually, markings dark enough → food source w/o random exploration*
- Probability bet uniform & high probability of trails
  - *For cells in immediate vicinity of nest since robot visits nest often*
  - *Therefore, important to emphasize trail (use "Sensing Areas High Density")*

Local Navigation: Obstacle Avoidance

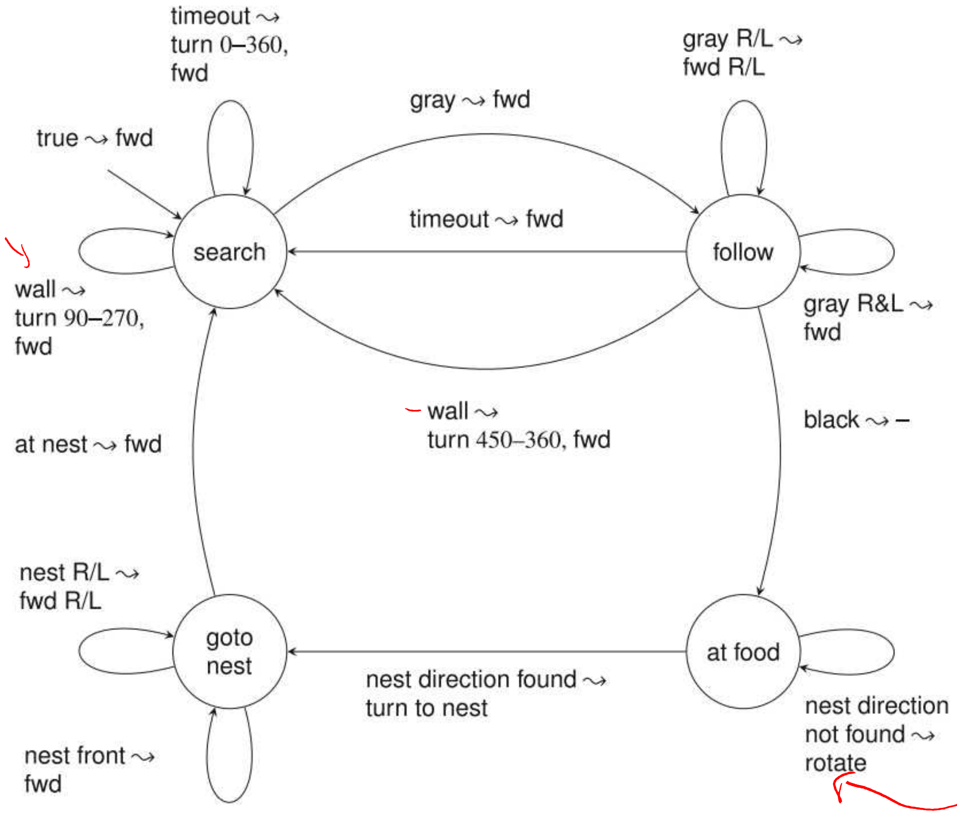
- Obstacle Avoidance
- Following a Line with a Code
- Ants Searching for a Food Source
- A Probabilistic Model of the Ants’ Behaviour
- A Finite State Machine for the Path Finding Algorithm

A FSM for the Path Finding Algorithm



State Machine for a drawing path between food source & nest.

A FSM for the Path Finding Algorithm



State Machine for a drawing path between food source & nest.

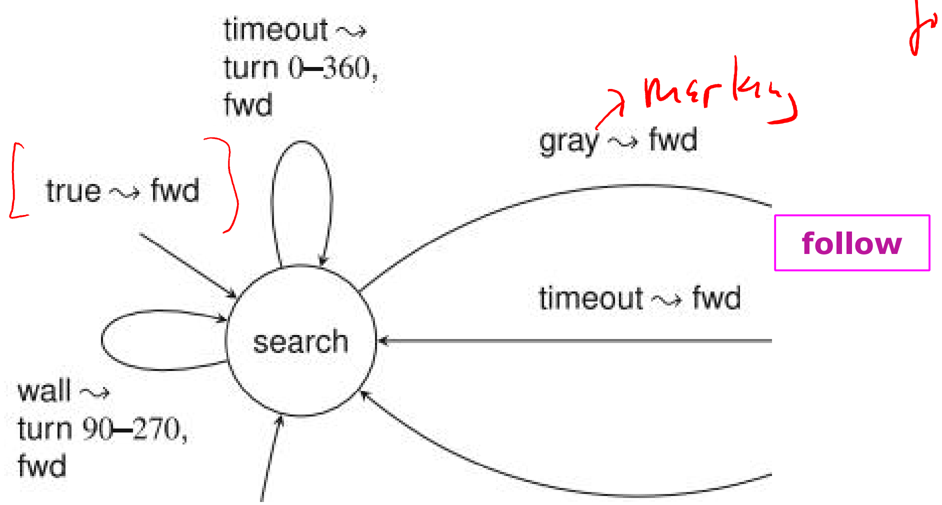
Item	Explanation
fwd	Set motor forwards
fwd R/L	Set motor forwards & to the right/left
	fwd & fwd R/L also set timer random period
Wall	Wall detected
Timeout	Timer period expired
Gray R/L/R&L	Gray detected by right/left/ both sensors
Nest front/R/L	Nest detected in front/right/left
Black	Black detected
Nest direction	Dir from food → nest found → not found
Turn $\theta_1 - \theta_2$	Turn randomly in the range $\theta_1 - \theta_2$
Rotate	Robot (or its sensor) rotates

Abbreviations in the state machine

A FSM for the Path Finding Algorithm

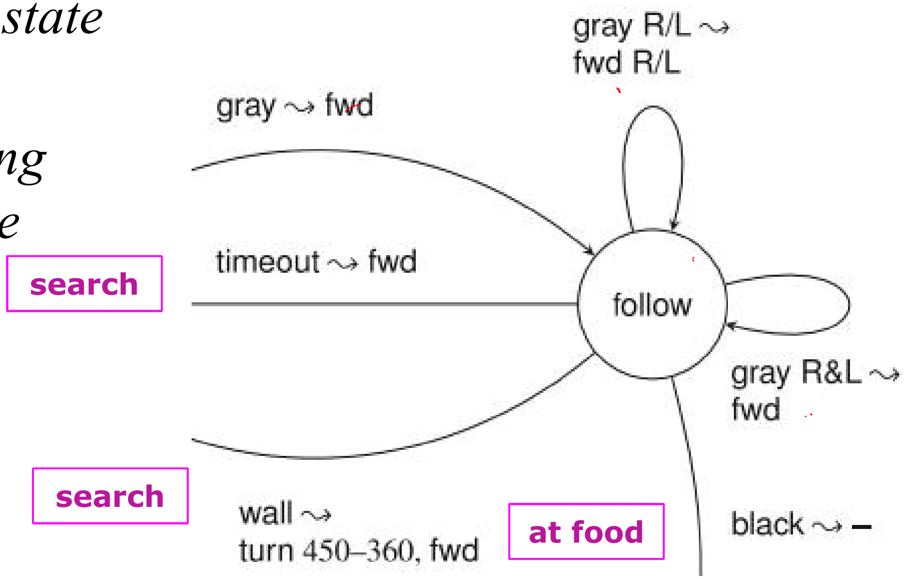
- search
  - initial *state; randomly search for dark areas*
  - *true ~> fwd* **:** *initially & unconditionally move forward & timer set random*
  - *timeout ~> turn 0°- 360°, fwd* **:** *random turn/ move forward & resets timer*
  - *wall ~> turn 90°- 270°, fwd* **:** *random turn from wall (sensor face fwd)*
  - *gray ~> fwd* **:** *transition to* **follow** *state*

fwd = black



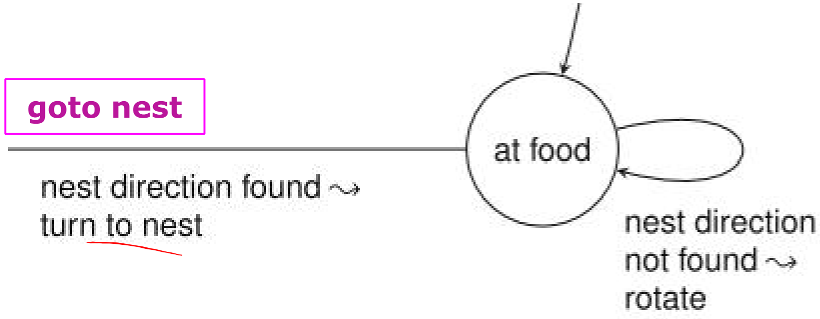
A FSM for the Path Finding Algorithm

- **follow**
  - *gray R/L ~> fwd R/L; gray R&L ~> fwd* **:** *line following implementations*
  - *timeout ~> fwd* **:** *timeout w/o detecting gray* → *robot not following line anymore* → *return to* **search** *state*
  - *wall ~> turn 450°- 360°, fwd* **:** *full 360° check for gray marking* → *turn away* → *return to* **search** *state*
  - *black ~> -* **:** *robot sense high-density marking (black)* → *conclude food source reached* → *transition to* **at food** *state*



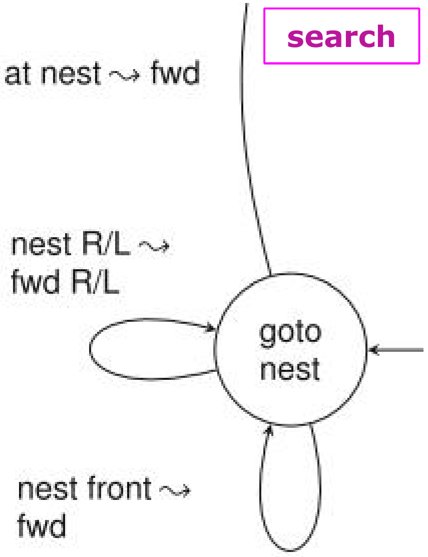
A FSM for the Path Finding Algorithm

- at food
  - *food source discovered → must return nest*
  - *nest can be detected but robot sensor not necessarily face direction of nest*
  - *nest direction not found ~> rotate* : *look for direction to nest*
  - *nest direction found ~> turn to nest* : *robot (sensor) rotate → find direction to the nest → turn towards nest → transition to* **goto nest** *state*

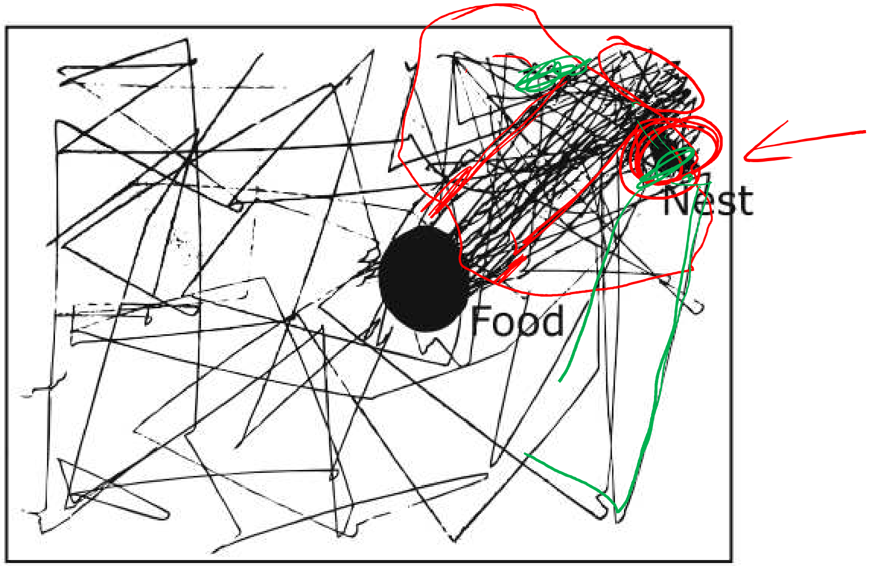

goto nest
at food
nest direction found ⇝ turn to nest
nest direction not found ⇝ rotate

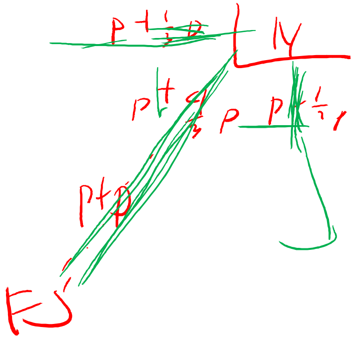
A FSM for the Path Finding Algorithm

- **goto nest**
  - *similar to* **follow** *state*
  - *at nest ~> fwd* **:** *move forward to nest* → *turn right/left as needed in direction of nest* → *transition to* **search** *state*
  - *nest R/L ~> fwd R/L* **:** *move toward direction of nest*
  - *nest front ~> fwd* **:** *move toward direction of nest*



Experimental Result





- High density of lines between nest and food source
- Also relatively high density in vicinity of nest
  - *Not necessarily in direction of food source*
- Can make robot go to random searching
  - *Instead of go directly to food source*

Summary

- Obstacle Avoidance algorithms
  - *Wall following algorithms in use since ancient times*
  - *Context of navigating a maze*
- Various anomalies can cause line following to fail
  - *Ex.: G-shaped obstacle traps the wall following algorithm*
  - *Pledge algorithm can deal with this*
- Colony of ants
  - *Determine path between nest and food source*
  - *Without knowing location & without a map*
  - *Reinforce random behavior with positive outcome.*

Thank you.