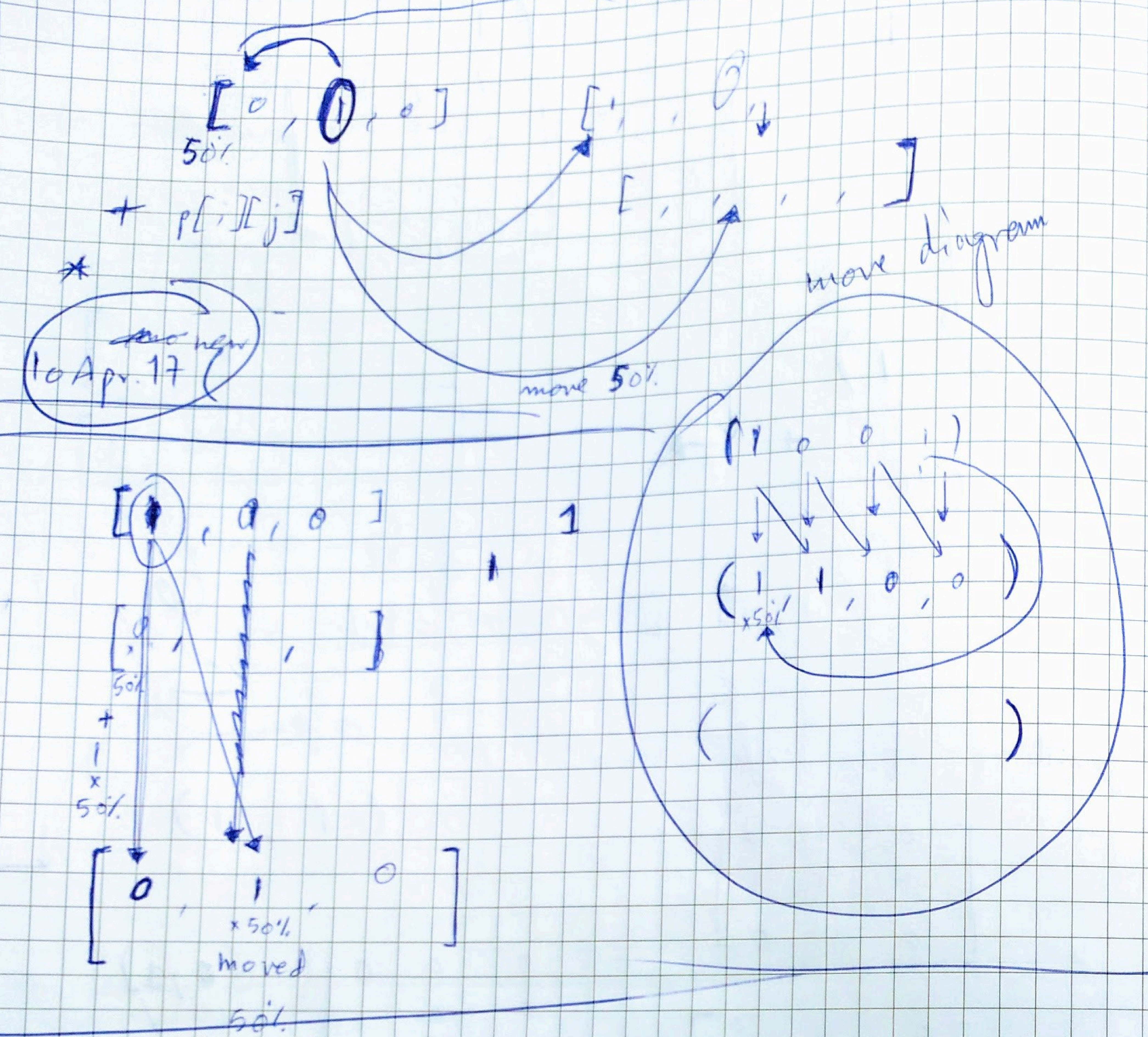


Algorithm for sense nxn

initial q with 0's, same dimension as p.

Loop:

if color[outer][inner] is measurement:

q(outer)(inner) = p(outer)(inner) * sense_right

otherwise

↓ *sense_wrong

Sum += q(outer)(inner)

Now, normalize the values:

for Loop:

q(outer)(inner) /= Sum

return q

Algorithm for move:

* initialize q with 0's, same dimension as p.

* Loop: outer / inner

move_value = p(outer)(inner) * p_move

not_move_value = p[motion_y % len(p)][motion_x % len(p(outer))]

q(motion_y)(motion_x) = move_value + non_move_value * p_stay

motion_x = (motion_x + 1) % len(p(outer))

motion_y = (motion_y) % len(p)

Office Hours:

→ localization depends upon having a map of the environment which is done using SLAM (simult. loc. & mapping) which covers the problem of acquiring the map at the same time

→ Currently self driving cars doesn't work in snow.

→ electro steering booster: to control steering using motor

→ lego minestorm

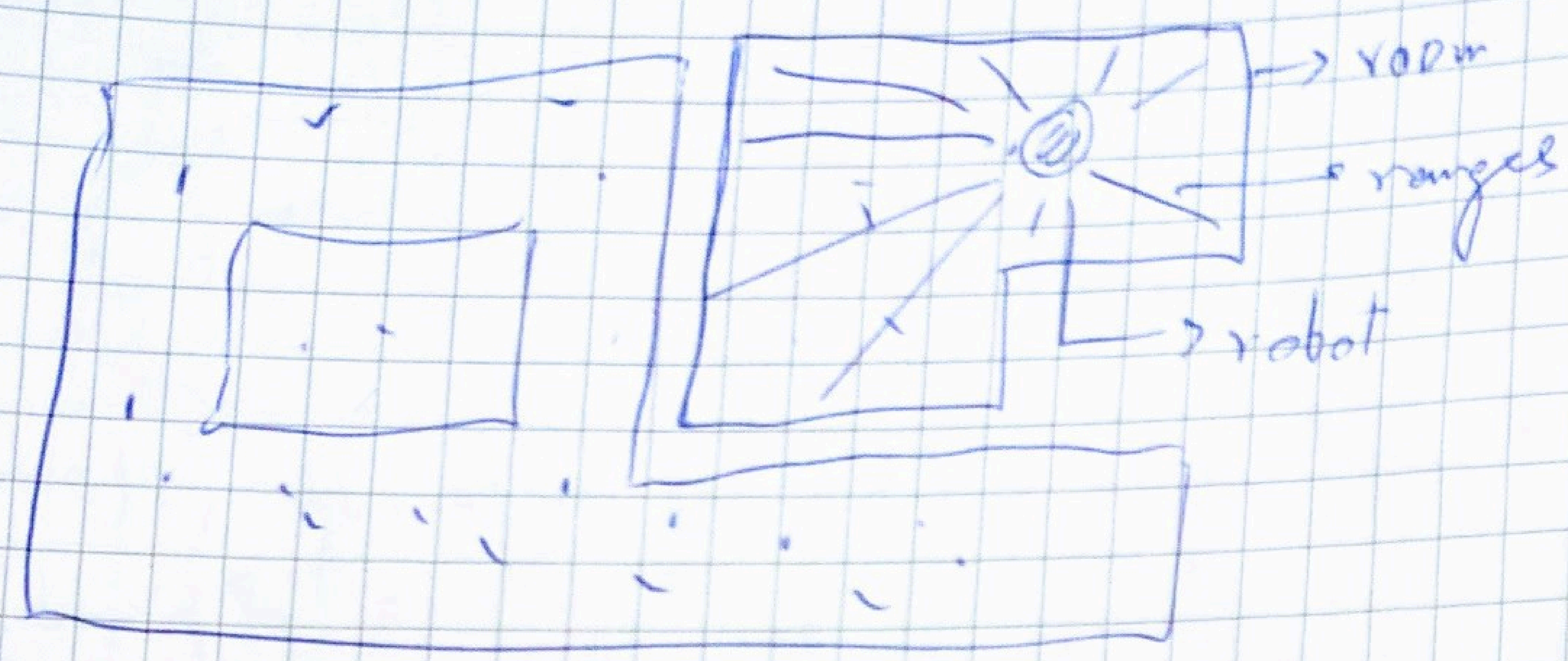
* Stanley had 4 [illegible] controller & radar. Could only see forward.

* Junior has 2 quad core CPUs & LIDAR & radar. Spinning one is LIDAR to provide info of 360.

* Bayes Rule is used to merged data of LIDAR & RADAR.

particle filter

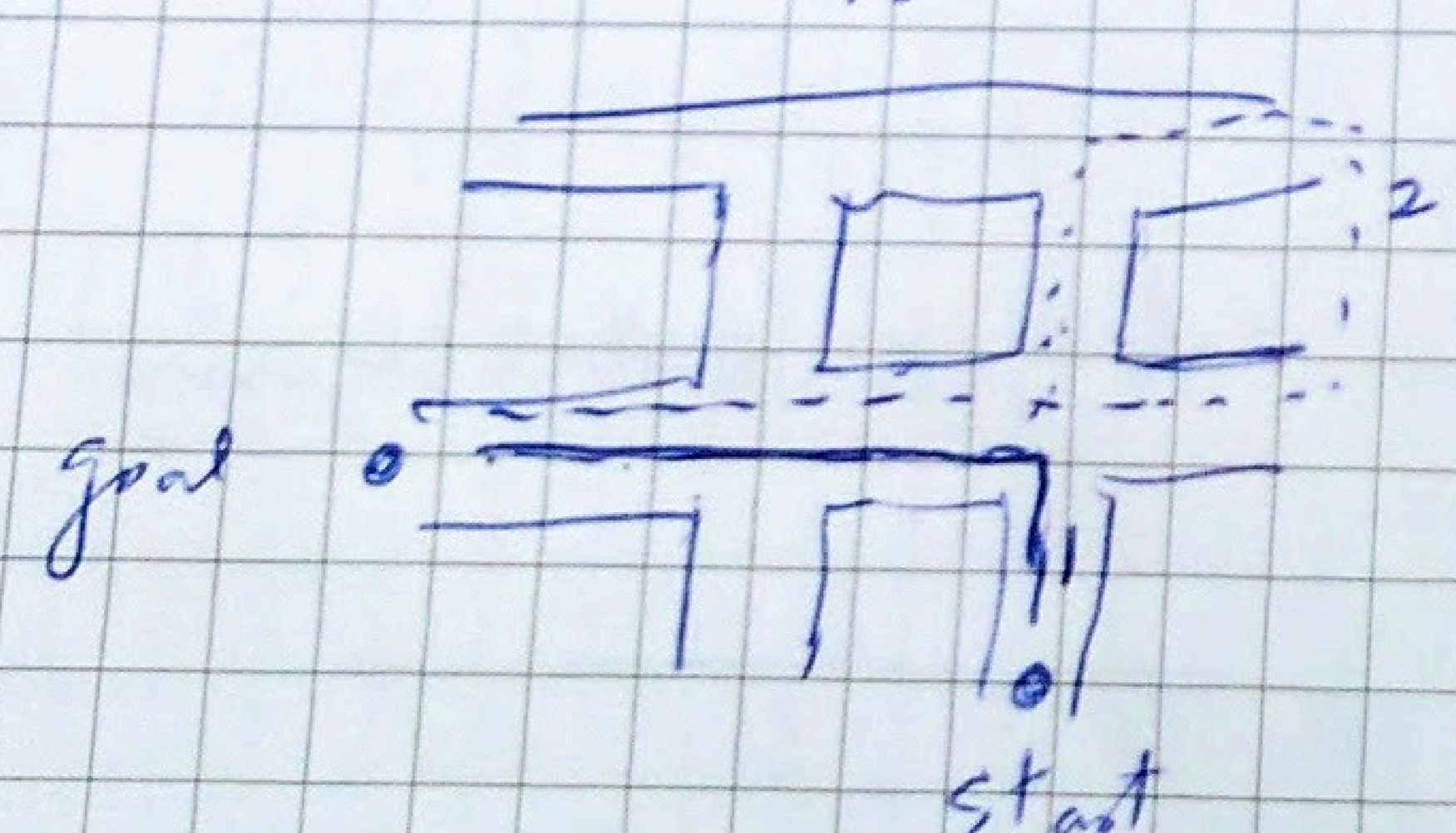
⇒ for sensing the range, SONAR is used



⇒ global localization:
- robot has no idea where it is, now based on sensors we need to localize it
- "particles" are discrete guesses
  ↓
  (x, y, θ)
  ↳ heading direction

~~Motion~~ Search.

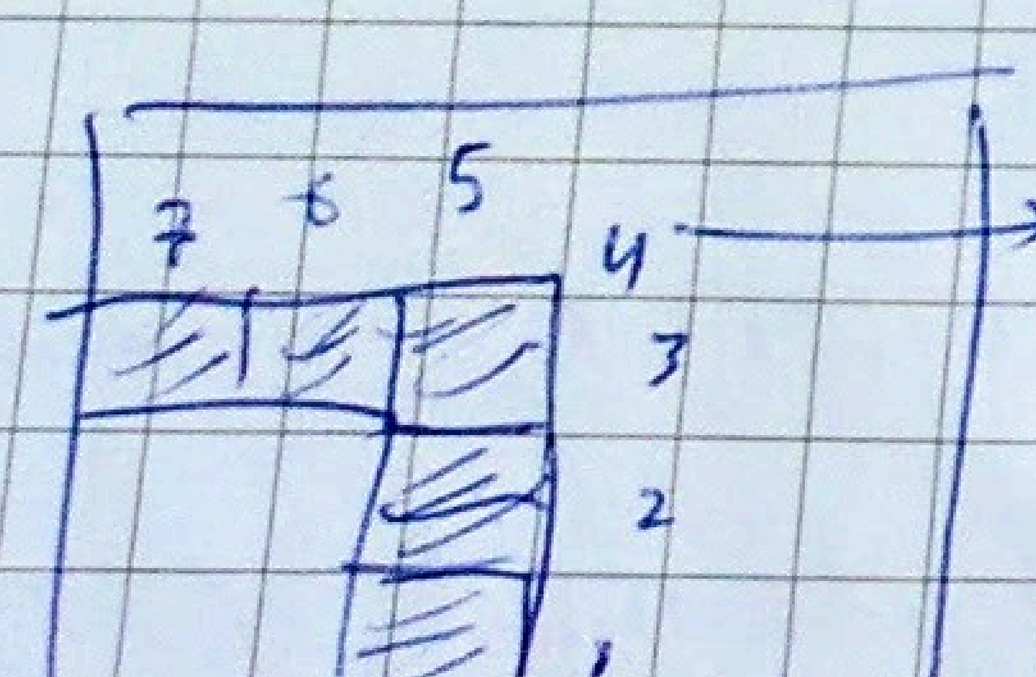
Robot Motion
→ Planning ⇒ process of finding path from start to required location.



given
- Map
- Start loc
- Goal
- cost

Required
- Min. cost path

Cost

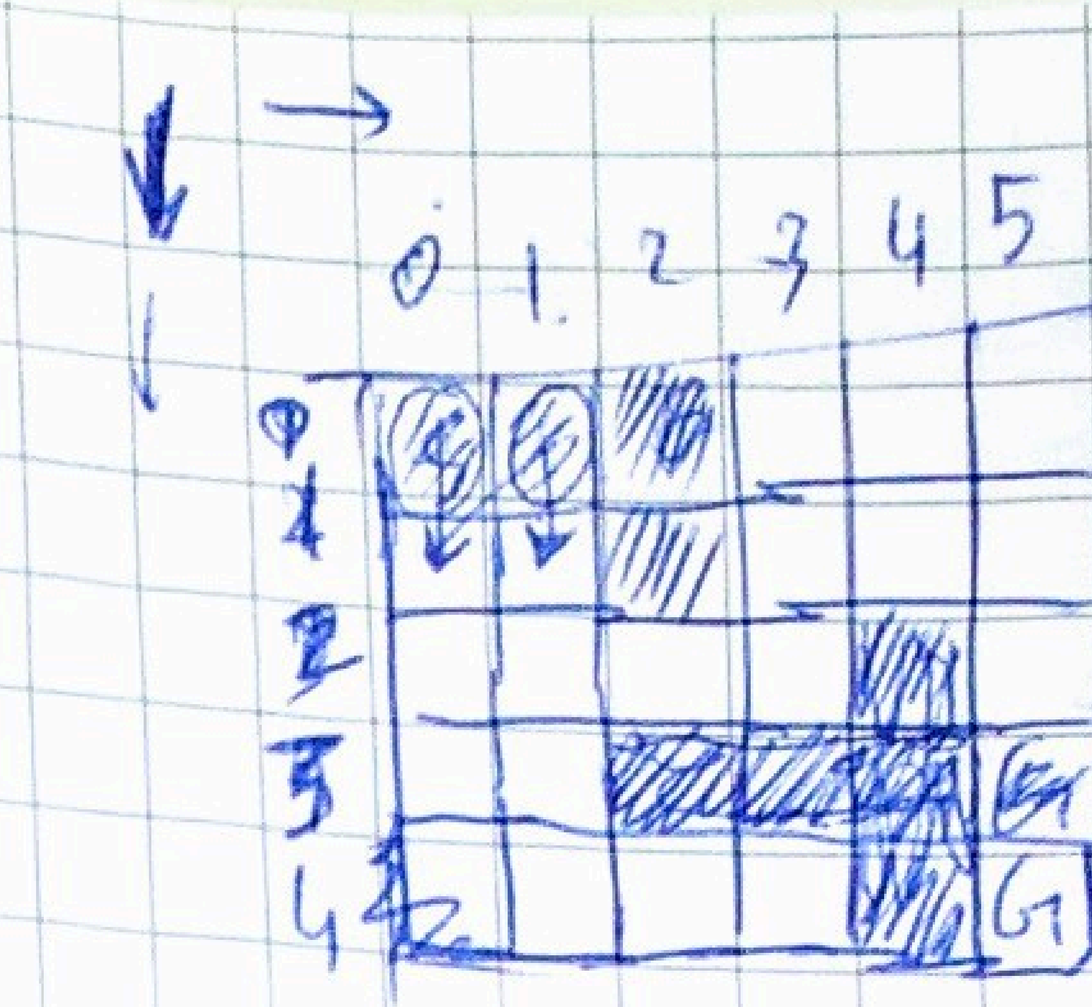


→ cost of turn
So.
6 → straight motions cost
1 → turn cost

- In real traffic, left turn has high cost b/c left turn are harder to do than right turn
- Fedex & ups tries ~~to~~ to avoid routes w/ left turns.

16

↗ Search program:



(row, col)
Successor 1 (row+1, col)
Successor 2 (row, col+1)

Open = [[0, 0]]

0, [gvalue, x, y]

	0	1	2	3	4	5
0	0	0	1	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	1	1	0
4	0	0	0	0	1	0

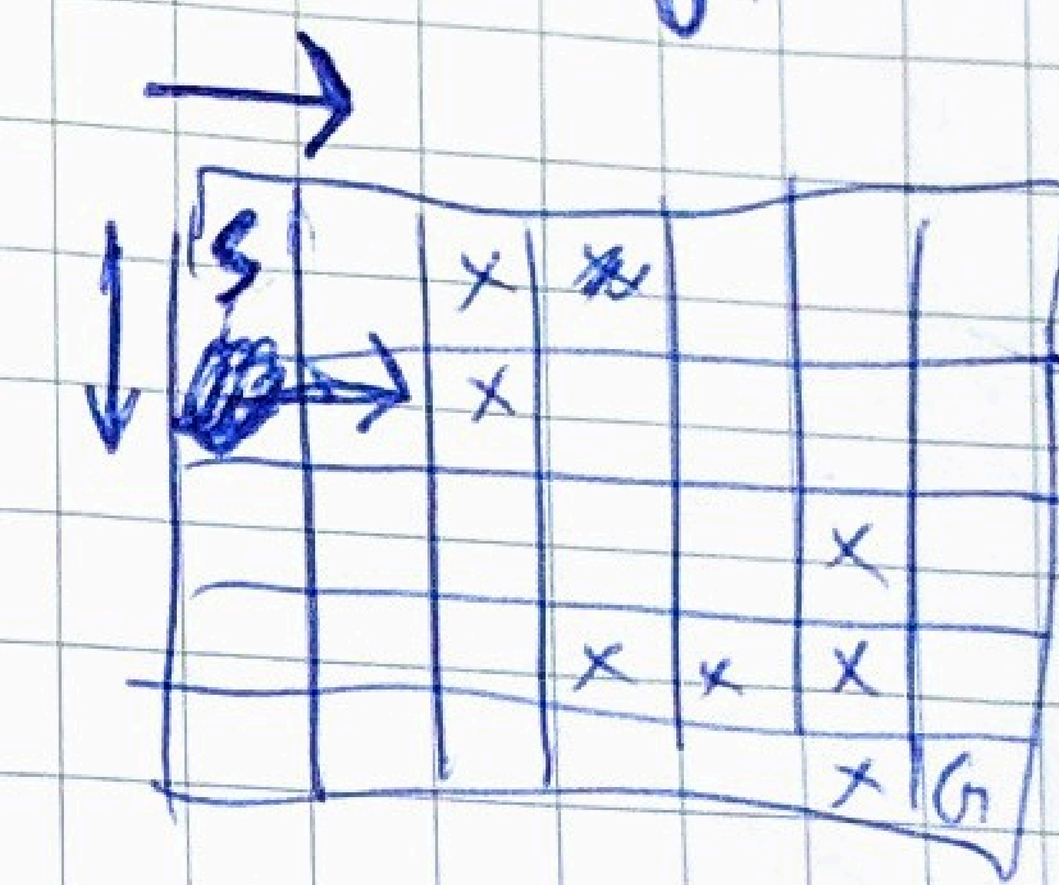
- Create open list
- remove element w/ smallest g value.
- If g value of 2 elems, ~~is~~ is same remove ~~any~~ element which has value 1.

[(0 0 1 0 0 0), (0 0 1 0 0 0),
(0 0 0 0 1 0), (0 0 1 1 1 0),
(0 0 0 0 1 0)]

Initial = [0, 0]

[1, 0, 1]
[1, 1, 0]

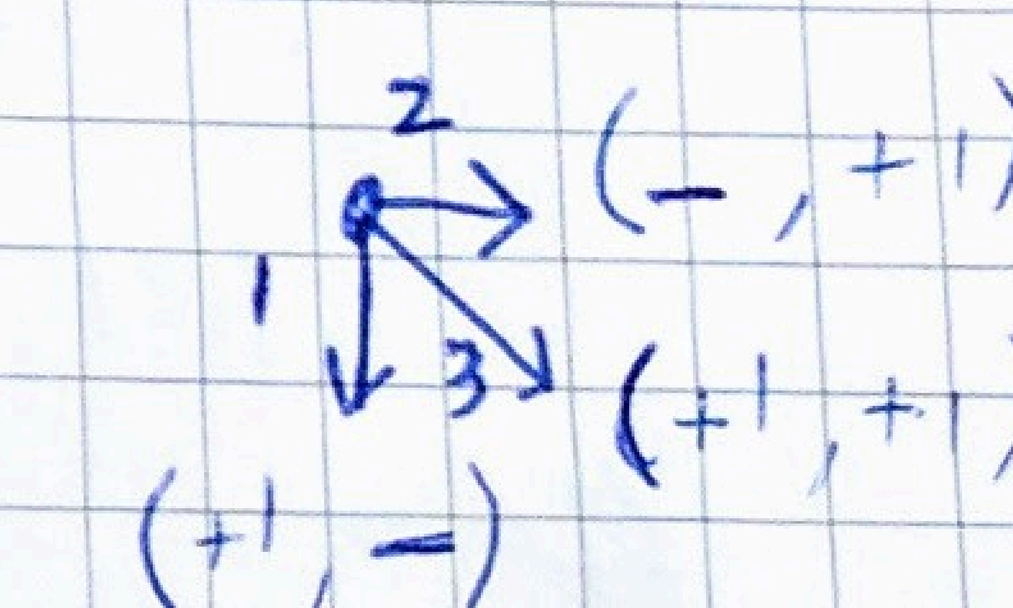
y, x



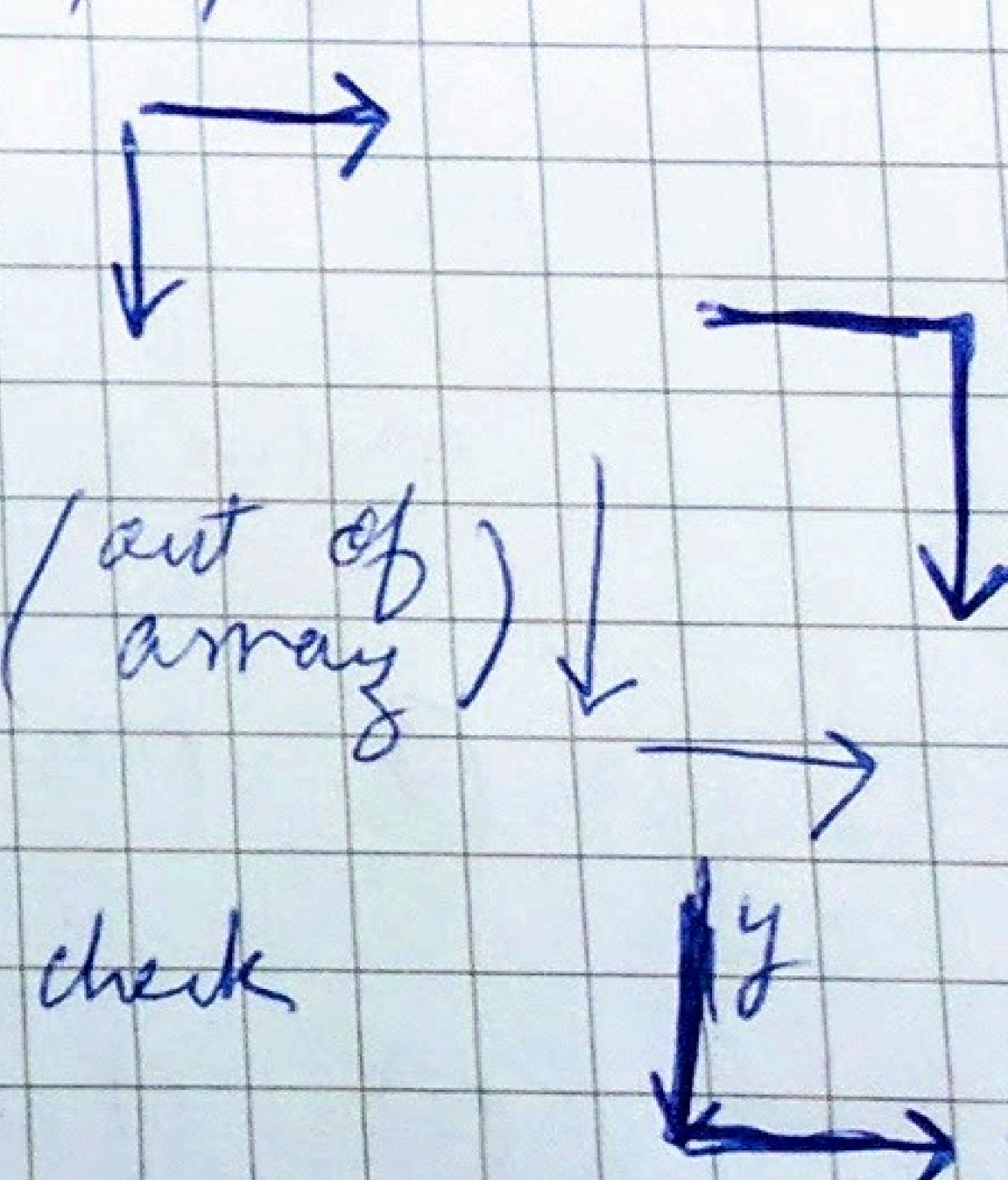
Take
[1, 0, 1]

(1, 0, 1)
4 (1, 1, 0)
(1, 0, 1)

(1, 1, 0)



- After every move, see if new one is already closed/
  "OR" if its value is not defined (out of array)
  "OR" if its blocked (has value 1)
  ↳ same as 1st check

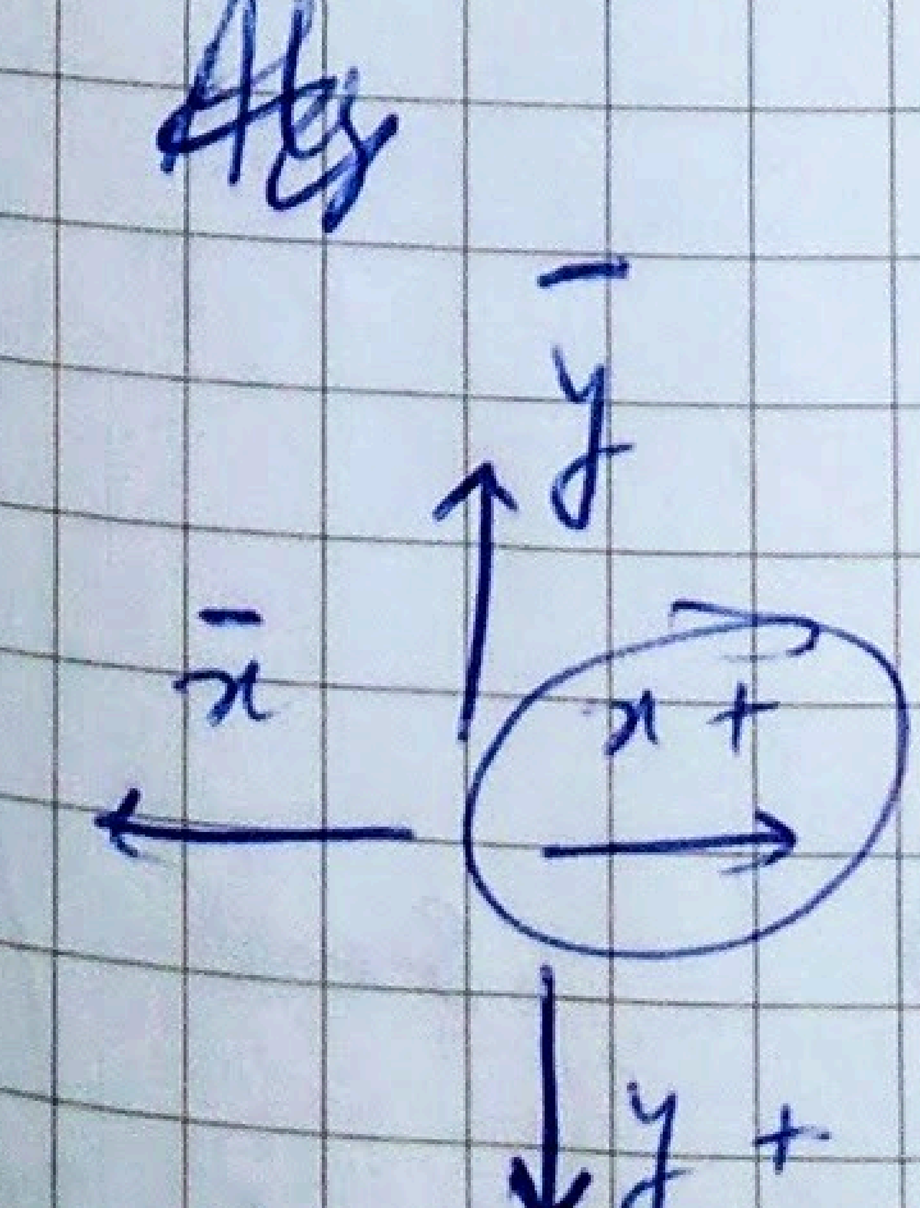


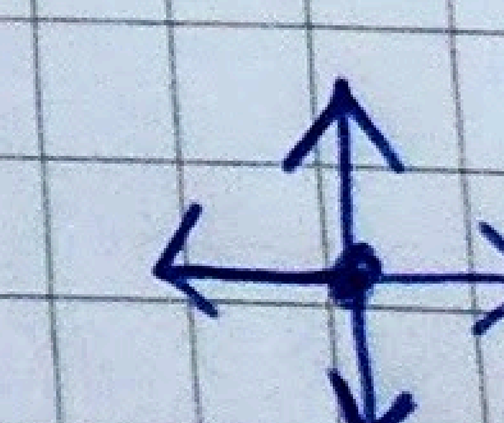
- goal = [4, 5]

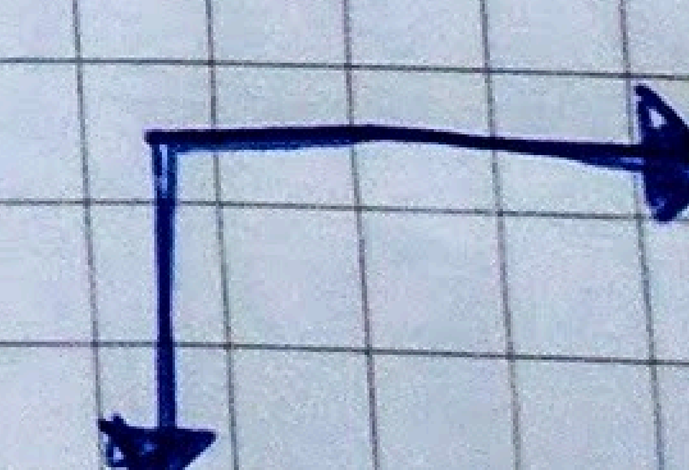
→ ~~if (left > −1)~~ →

if (x+1)(y) > −1 →
elif (
(x)(y)

Algo







Algorithm:

① - Create empty closed array ; same dimension as original.

② - mark it closed

③ - open = [[g, x, y]]

where g = 0 ; x = init[0] ; y = init[1]

while (found = false && resign is false):

if : open list is empty i.e length is 0

resign is true

otherwise :

① - remove node from open list

* reverse sort "open". It will give you list in decreasing order.

② - now pop the element. This will give the last element of open, which has smallest value. Save it in "next"

③ - Now x = next(1) ; y = next(2) ; g = next(0)

④ - check if x, y matches to goal coordinates then set found as true

⑤ : If not done :

run loop over delta

* new x2 is x + delta(i)(0) && same for y ; inside of grid

* check if x2 is positive/zero and its length is < grid's and y2 " " " " " "

* Check if closed(x2)(y2) is 0 && grid(x2)(y2) is 0

* g2 is g + cost

* append open with [g2, x2, y2]

* x2, y2 element of closed set as 1 ; i.e closed.

* Algorithm again.

* Init empty closed with same dimensions as grid.

* g = 0 ; x = init(0), y = init(1), open = [[g, x, y]], cost = 1

* closed(init(0))(init(1)) = 1 # closed.

* resign = false, found = false

* while resign is false and found is false:

if len(open) is 0:

resign = True

else:

open.sort() ; open.reverse() ; next = open.pop()

g = next(0) ; x = next(1) ; y = next(2)

if (goal[0] is x and goal[1] is y):

found = True

else:

for index in range(len(delta)):

x2 = x + delta[index][0]

y2 = y + delta[index][1]

if x2 >= 0 and x2 < len(grid)(0) and

y2 >= 0 and y2 < len(grid)(0):

if grid(x2)(y2) is 0 and

closed(x2)(y2) is 0 :

g2 = g + cost

open.append([g2, x2, y2])

closed[x2][y2] = 1

if found is True :

return [g2, x, y]

else : return "fail"

→ Draw the path of the robot

Algo:

loop over row & col

~~access~~ if ~~current~~ next value is 1 (or 0)

increment row

mark current as 'V'

else

mark current as '>'

Algorithm for path tracing:

* save the index of delta in action matrix at the time when you loop over delta in path finding.
* mark goal-th element as '*'.
* Now move back
* set x and y as goal x & y
* loop until x & y as init x & y
* we took next position in finding algo using

$x2 = x + delta(index)(0)$

$y2 = y + delta(index)(1)$

* here since we reversing back, we'll do:

$x2 = x - delta(action(x)(y))(0)$

$y2 = y - delta(action(x)(y))(1)$

$x = x2$

$y = y2$; recursion

$path(x)(y)(as) = delta\_name(action(x)(y))$

A* Method

* In normal expansion grid:

open (0,0), g

* In ~~Heuristic~~ A# expansion grid:

open (0,0), g, g+h(x,y)

and

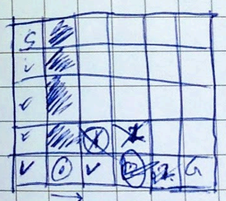
$f = g + h(x,y)$

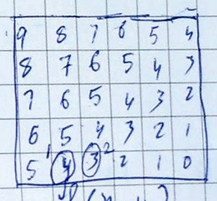
So here:

open (0,0); 0, 0+9

Now during expansion, we'll remove element with Lowest 'f' value & not the Lowest 'g' value

Consider:





h(x,y)

1- open [0,0], 0, 9

2- ~~open~~ [1,0], 1, 1+8 [1,1], 1, 1+8 [2,0], 2, 2+7

3- ~~open~~ [1,1], 1, 1+8 [2,0], 2, 2+7 [2,1], 2, 2+7 [3,0], 3, 3+6

$f = g + h(x,y)$

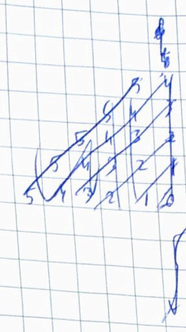
4- [2,0], 2, 2+7 [2,1], 2, 2+7 [3,0], 3, 3+6 [3,1], 3, 3+6 [4,0], 4, 4+5

5- [2,1], 2, 2+7 [3,0], 3, 3+6 [3,1], 3, 3+6 [4,0], 4, 4+5 [4,1], 4, 4+5

6- [3,0], 3, 3+6 [3,1], 3, 3+6 [4,0], 4, 4+5 [4,1], 4, 4+5 [3,2], 7, 7+4

7- [3,1], 3, 3+6 [4,0], 4, 4+5 [4,1], 4, 4+5 [3,2], 7, 7+4 [3,3], 8, 11 (8+3)

* In A*, heuristic function table providing the additional info to reduce expansions



9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

* $h(x,y) \leq$ dist. to goal
* This $h(x,y)$ from x,y is used in freeform navigation
* In reality, we don't measure # of grids to the target but the euclidean dist. to a target location
* If $h(x,y) = 0$ Then it falls back to normal expansion grid method.

//// → means this node is expanded

9 is here is small, so we'll choose it

[4,3], 4, 7+2

here, its 1st time A* makes the difference

[4,4], 8, 9 (8+1)

Refused

* grid:

0	0	1	0	0	0
1	0	1	0	0	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0

using expansion grid:

0	-1				
1	-1				
2	-1				
3	-1				
4	-1			-1	
5					

using Exp. Grid Method:

0	-1	14	18	21	23
1	-1	11	15	19	22
2	-1	9	12	16	20
3	-1	7	10	13	17
4	5	6	8	-1	24

with heuristic

	0	1	2	3	4	5
0	9	8	7	6	5	4
1	8	7	6	5	4	3
2	7	6	5	4	3	2
3	6	5	4	3	2	1
4	5	4	3	2	1	0

	0	1	2	3	4	5
0		1	0	0	0	0
1		1	0	0	0	0
2		1	0	0	0	0
3		1				0
4					1	0

open

[0,0], 0, 9

[1,0], 1, 9 (1+8)

[2,0], 2, 9

[3,0], 3, 9

[4,0], 4, 9

[4,1], 5, 9 (5+4)

[4,2], 6, 9 (6+3)

[3,2], 7, 11 (7+4)

[2,2], 8, 13

[3,3], 8, 11

[2,3], 9, 13

[3,4], 9, 11

[3,5], 10, 11

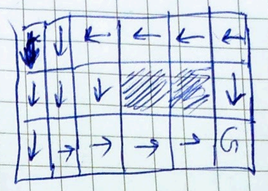
[4,5], 11, 11

* Dynamic Programming for Robot Path Planning

Given:
1- Map
2- Goal

Outputs:
Best path from Anywhere

every point can be treated as startpoint



* policy
A function which maps grid cell to action
(x, y) → action (more ↑ ↓ ← →)

* Stochastic Environment
→ outcomes of actions are non-deterministic

* dynamic programming gives optimal action to do at every single gridcell
* every cell has a label called policy

(1) Skimming Kalman filters — Sebastian Thrun

LASER Range finder, gps

used for tracking, similar to Monte Carlo

Localization

Kalman
- Continuous
- Uni Modal
- Gaussian Distribution

Particle filters
- Continuous
- Multi Modal (Histogram)

Monte Carlo
- Discrete
- Multi Modal
- Normal Distribution
- Histogram

* Markov Modal

Prob. 0.2, 0.1, 0.1, 0.5 → Space

So:

1D Gaussian: M (mean), σ^2 (variance), x

→ Normal Distrib. (Monte Carlo)

→ Kalman filter (Gaussian)

Continuous
- area is 1 (represents probability)

So, to attain true histogram we use Gaussian distribution (M, σ^2) for estimation

$$f(x) = \exp -\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$$; x → Random Variable

↳ for normalization

when $x = \mu$

small σ^2 → Gaussians

↳ medium σ^2

σ^2

* Single Peak
* Expo * Unimodal

→ * Bimodal

large σ^2 So: Large σ^2 means $f(x)$ matters Lesser

So lesser peaks & more uncertain is System

→ high Confidence

→ low Confidence

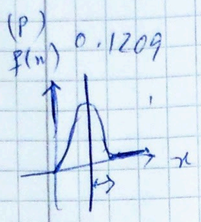
→ low probability
→ High σ^2

→ b/c high prob.
→ Low σ^2

$a^2-2ab+b^2$

* Gaussian is used as "belief" in Kalman

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{\frac{-1}{2}\frac{(x-M)^2}{\sigma^2}}$$



$\mu = M$
$\sigma = \frac{1}{2\pi}$

$$1 = \frac{1}{\sqrt{2\pi(4)}} e^{\frac{-1}{2}\frac{(x-10)^2}{4}}$$

$$0.19947 = e^{\frac{-1}{8}(x-10)^2}$$

$\log(e^8)$

$\ln(0.19947) = -\frac{1}{8}(x-10)^2 \ln(e)$

$\ln e^x = \ln 55$
$x \ln(e) = \ln 55$

$-1.612 = -\frac{1}{8}(x-10)^2$

$12.896 = (x-10)^2$

$x - 10 = 3.59$

$\boxed{x = 13.59}$

$x = 24.59$

$x = -4.59$

$$\frac{1}{\sqrt{2\pi(4)}} e^{\frac{-1}{2}\frac{(13.59-10)^2}{4}}$$

$$\ln\frac{0.199}{\sqrt{2\pi\ 4}} = \ln\left[e^{\frac{-1}{2}\left(\frac{(x-10)^2}{4}\right)}\right]$$

$$\ln\left(\sqrt{2\pi 4}\right) = \frac{-1}{2}\frac{(x-10)^2}{4}\ln(e)$$

$-8(1.612) = (x-10)^2$

$-12.8966 = x^2 - 20x + 100$

$x^2 - 20x + 112.8966 = 0$

① $f(x)$ is max. when $x = \mu$ (rand. variable = mean)

for $f(x) = 1 \Rightarrow x = M$ & $\sigma^2 = \frac{1}{2\pi}$

* Kalman filter

measurements (update) → requires product → updating belief by multip. factor

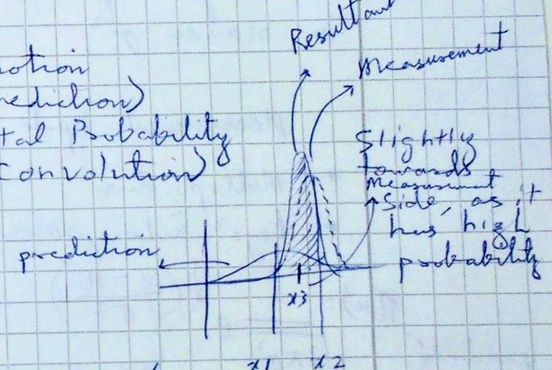
motion (predict) → requires convolution

* In measurement, we update our belief using Baye's Rule (apriori & apostariori) (mmse)

in motion, we keep track of where total probability went when we moved, using Law of total probability.

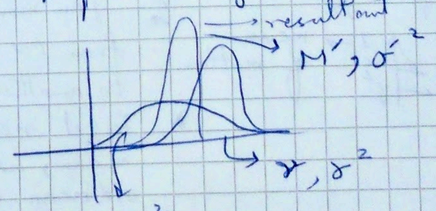
Measurement (Update) Bayes Rule (Product)

Motion (prediction) Total Probability (Convolution)



* More measurements means greater certainty. &
↑ probability ↓ Covariance

* ↳ And the resultant probability will be higher than both prediction & update measurements have lesser Covariance



So:

$\Rightarrow M' = \frac{M(r^2) + V(\sigma^2)}{r^2 + \sigma^2}$ → weighted sum

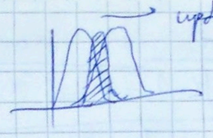
$\Rightarrow \sigma'^2 = 1/(1/r^2 + 1/\sigma^2)$ → unaffected by means

} updated results

$\mu = 10 \; ; \; \sigma^2 = 4$

$\nu = 12 \; ; \; r^2 = 4$

$\mu' = 11 \;\&\; \sigma'^2 = 1/\left(\frac{1}{4} + \frac{1}{4}\right) = 2$



$\mu = 10 \; ; \; \sigma^2 = 8$

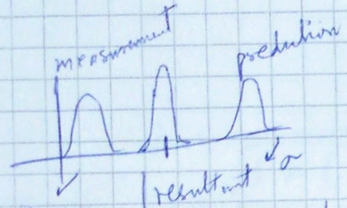
$\nu = 13 \; ; \; r^2 = 2$

$\mu' = 12.4 \; ; \; \sigma'^2 = 1.6$

b/w μ & ν

↳ less than σ^2 & r^2 which means high probability

on each update, probability increases



As:

$$\sigma'^2 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{r^2}} \Rightarrow \frac{\sigma^2}{2} = \text{less than } \sigma^2$$

It's b/c we got more information than anyone of them, it results in narrow σ^2.

Measurement Update

* Multiplication
* Bayes's Rule

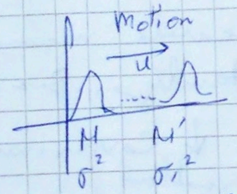
$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)}$$

(mean; Random Value; variance (confidence); probability)

1)

Motion Update

* Prediction
* Convolution (Addition)
* Total Probability



$\mu' = \mu + u$

$\sigma'^2 = \sigma'^2 + r^2$

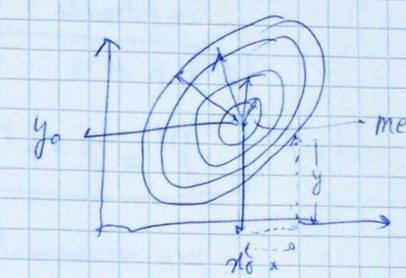
So you moved from one loc. to another.

* To get new mean, add old mean w/ motion mean.
* To get new variance, add old variance w/ motion variance

---

③ - Multivariate Gaussians for Kalman Filter

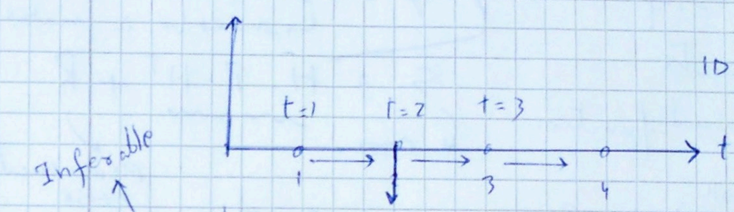
(i)

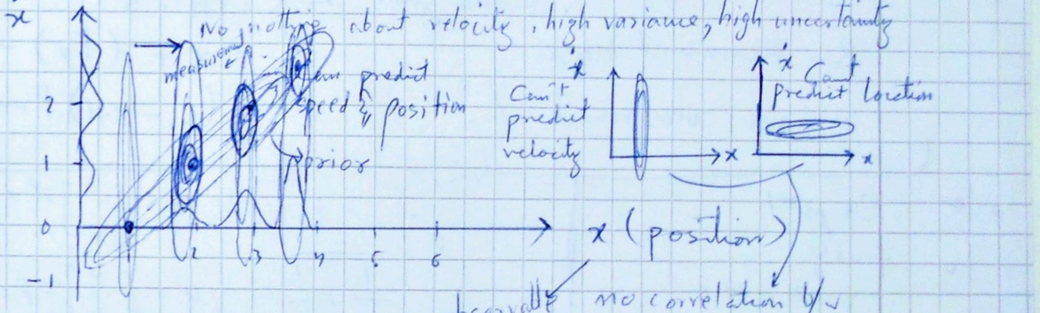


2D Gaussian

- mean is repres. by Contours.

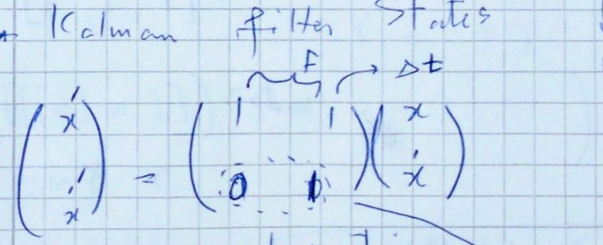
- Here x & y are correlated, so if I have info about x then I have estimate for y.





$x' = x + \dot{x}t$ ← old position; ← time; ← velocity; ← moved distance; ← new position

- Kalman filter States



$$\begin{pmatrix} x' \\ \dot{x}' \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

$$z = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

(3)(ii)

Kalman Filter In Higher Dimension

⇒ prediction Step

x' → new estimate
x → estimate
F → State transition matrix
u → motion vector
P → Uncertainty Covariance

$$x' = Fx + u$$
$$P' = FPF^T$$

new estimate

⇒ Measurement Step:

z → measurements
H → measurement function
I → Identity Matrix
R → Measurment noise
y → new measurement

new measurement
Projecting System Uncertainty into measurement Space

$$y = z - H \cdot x$$
↳ should be in matrix form

$$S = H \cdot P \cdot H^T + R$$
(error
measurement noise

Kalman Gain ← $K = PH^T \cdot S^{-1}$

$x' = x + (Ky)$ → Updated Estimate

$P' = (I - K \cdot H) \cdot P$ → Updated Uncertainty

x, y → r, θ

Steps =1
finding
$y = z - Hx$
find system error
$S = H \cdot P \cdot H^T + R$
find gain
$K = PH^T \cdot S^{-1}$
find new estimate
$x' = x + K \cdot y$
find new Uncertainty
$P' = (I - KH) \cdot P$

Step - 2:
$P' = FPF^T$
$x = Fx + u$

$$\sigma' = \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

$$\mu' = \frac{\sigma_1^2}{\sigma_2^2}$$

$$F = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$$x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; P = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$$ high position Uncert.

$$u = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$ high Uncert. velocity

$H = [1 \quad 0]$ → Can observe position but not velocity

$R = [1]$

$I = [1 \quad 1]$

$z = [1, 2, 3]$

matrix([[m[n]]])

$x = Fx + bu$

$$A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$

$x = Fn + u$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

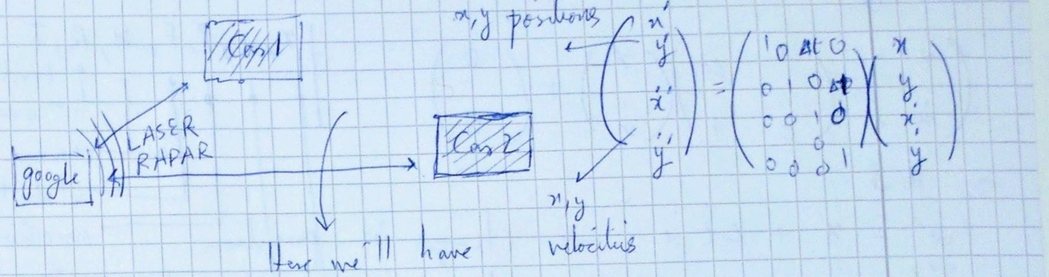
$$F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2

3 [2.398

2 [0.999

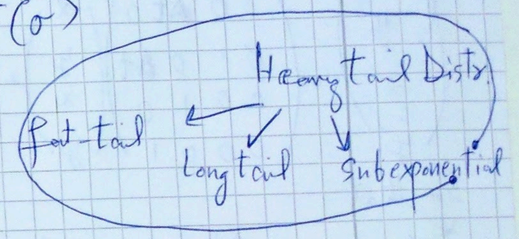
(4)



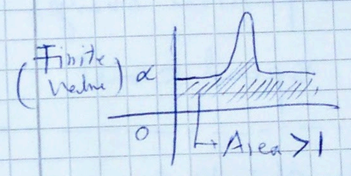
Correspondent problem where we don't know the data is from which

*

$\sigma_R^2 = 0.5(\sigma^2)$



* Heavy tail Guassian



Its not possible to represent tail guassian with:

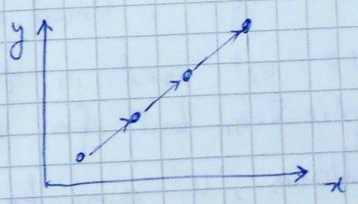
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)}$$

B/c

for $x \to \infty$ $f(x) = 0$ for guassian b/c $e^{\infty} = \infty$ $e^{-\infty} = 0$

* Its invalid probability distribution b/c area under the curve is > 1 (it diverges) in case of HiTail guassian

* $\begin{pmatrix} x \\ \dot{x} \end{pmatrix}$ → 1 dimensional
x dim. pose
$\dot{x}$ dim. velocity



(see previous diagram like Th.1, but it was for x & $\dot{x}$)

* $\begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix}$ → 2 dimensional

for 1D:

$$\begin{pmatrix} x' \\ \dot{x}' \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

So that:

$x' = x + \Delta t\, \dot{x}$ → change in position

$\dot{x}' = \dot{x}$ → Const. speed.

x, u, P, F, H, R, I

- I → identity
- R → meas. noise
- H → meas. matrix
- F → state tr. matrix
- P → uncertainty Cov. Matrix
- x → estimate matrix
- u → motion vector

* Kalman filter assumes constant velocity

for 2D:

$$\begin{pmatrix} x' \\ y' \\ \dot{x}' \\ \dot{y}' \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

So that:

$x' = x + \Delta t\, \dot{x}$; $y' = y + \Delta t\, \dot{y}$ → position changes

$\dot{x}' = \dot{x}$; $\dot{y}' = \dot{x}$ — Speed const.

→ Kalman f: Here 2-dimensional → 4 state vectors

* $z = [[5, 10], [6, 8], [7, 6], [8, 4], [9, 2]), (10, 0)$

$$x = \begin{pmatrix} 4 \\ 2 \\ 0 \\ 0 \end{pmatrix} ; u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$y = \frac{s}{t}$

$y = \frac{d}{dt}(s)$

* $P = \begin{pmatrix} \text{Uncert. of } x & 0 & 0 & 0 \\ 0 & \text{Uncert. of } y & 0 & 0 \\ 0 & 0 & \text{Uncert. of } \dot{x} & 0 \\ 0 & 0 & 0 & \text{Uncert. of } \dot{y} \end{pmatrix}$

* u → motion vect. → It is not used in tracking application but for actuation problems, 'u' can be used to manipulate motion.

5

* R → represents noise in the sensors. In case of white noise. It's 0.
* Radar also measures how fast things are moving away/towards using doppler effect.
* LASER (Lidar) has higher spatial resolution
* Bayes Rule is used to complement results of both.
* Surrounding Sensing → when RADAR can look in all directions.
* Spinning obj. in Self Driving Car is Lidar, it scans 360°.

* 2D kalman filter:

dt = 0.1

$$x = \begin{pmatrix} -4 \\ 8 \\ 0 \\ 0 \end{pmatrix} ; P = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 1000 \end{pmatrix} ; u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$F = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} ; H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} ; R = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}$$

$z = [(1, 4), (6, 0), (11, 4), (16, 8)]$

In python

z = np.matrix([z[index]]) ; → transform lists element to matrix

* AWGN → Adaptive White Gaussian Noise ⇒ Noise components are uncorrelated.

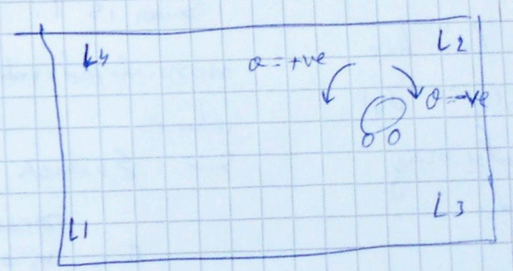
⇒ Particle filters.

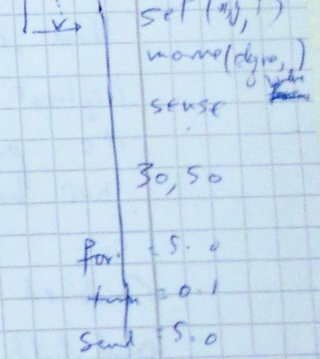
- Histogram filter scales exponential while Kalman filter scales quadratically.
  So kalman filter is more efficient

- Kalman & Histogram both are approximate filters & not exact.
  Kalman is exact for linear system but approx.
  For Non-linear (all real world cases).

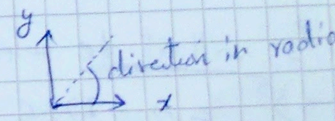
- Particle → Continuous, Multimodal, Aposon.
  → should'nt be used for more than 4 dimensions.

- SONAR → Sound based sensors — Sound Navigation & Ranging
- LIDAR → LASER based. — Light Detection & Ranging.
- RADAR → Radioware based — Radio Detection & Ranging



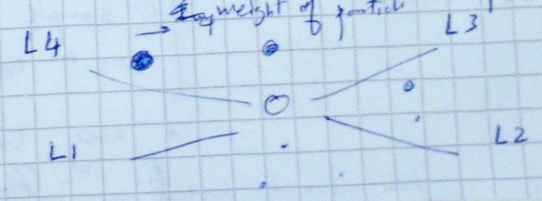


⇒ ⊗ N = 1000 ⊗ → (x, y direction)
⊗ ⊗
↳ each of the dots is 3x1 vector



turn by 0.1
move by 5

⇒ mismatch b/w predicted & actual
" measurement is Called Importance Weight



Higher weight particles Survive

⇒ Selecting particles w.r.t to their weights so
↳ Resampling (in the end only high probability particles remain.

* So we move a robot, we get the landmark value, called measurement. Then we get the probability of these measurements. Then by resampling we discard all the measurements w/ low probability

⇒ Resampling

Here, even though 'P3' has highest probability. But it's possible that it may not be selected.

So:
Combined prob. of all others = 0.6
Probability that they all will be selected = $(0.6)^5$

And:
Prob. that P_3 will not be selected
$= (0.6)^5 = 0.07776$

or
It's 7.7% probable/chance that P_3 won't be selected.

* transational noise
* rotational noise
* measurement noise

robot.sense( )

* Normalization
↳ Adjusting range of values so that their Sum is 1.

normalized_value = $\frac{\text{value}^{1\cdots 2}}{\text{Sum}}$

where
Sum = Σvalues

	measurement	Normalized measurement
P_1 –	$w_1 = 0.6$	$\alpha_1 = 0.1$
P_2 –	$w_2 = 1.2$	$\alpha_2 = 0.2$
P_3 –	$w_3 = 2.4$	$\alpha_3 = 0.4$
P_4 –	$= 0.6$	$\alpha_4 = 0.1$
P_5 –	$= 1.2$	$\alpha_5 = 0.2$
	$\Sigma = 6$	$\Sigma = 1$

* prob. that of

* Program to takes lists of particle & weights & rearrange or sample particle w.r.t weights.

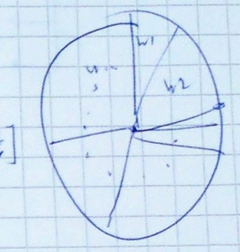
* use Sot Command

⇒ Resampling wheel Algorithm:

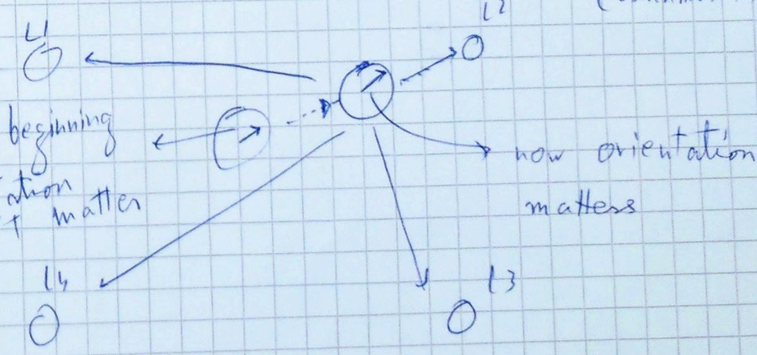
* It reorders the particles w.r.t their probabilities.

Algorithm:
w → weights of particle (array)
index = random(0 — N) [int]
beta → 0
for (i ... N):

```
beta = beta + random() * 2 * max(w)
while w(index) < beta{
    beta = beta - w[index]
    index = (index+1) % N
}
new_p[].append(p(index))
```



* Orientation of a particle is of no significance when its equidistance from all locations (landmarks)



* So orientations

⇒ Measurement update:

$P(x|z) \propto P(z|x)\ P(x)$ ↳ distribution
correct posterior ← resampling ↑ importance weights ↳ particles

Motion Update:

correct distribution after robot motion = new particle

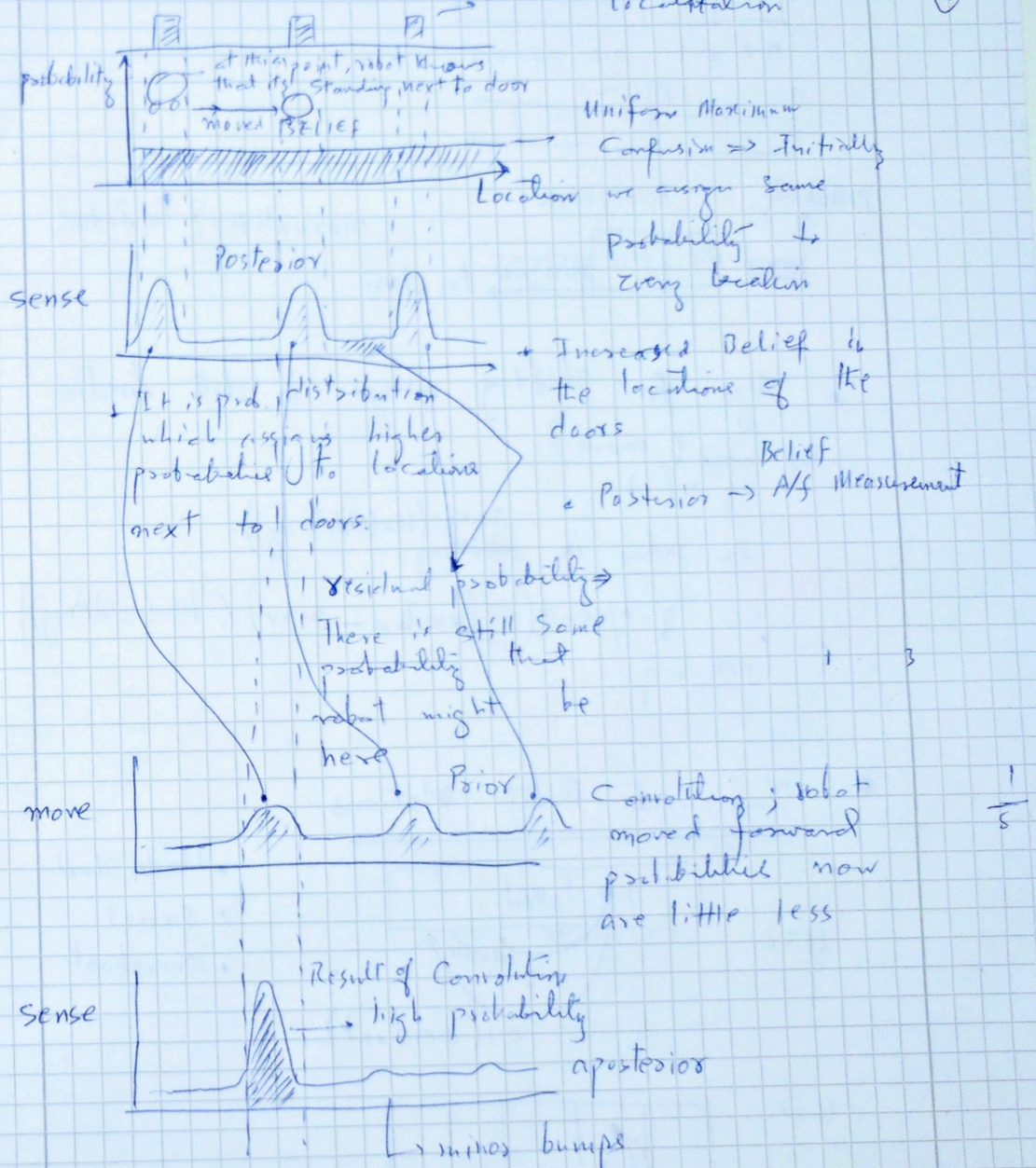
$P(x') = \sum P(x'|x)\ P(x)$
sample ↳ particles
motion model

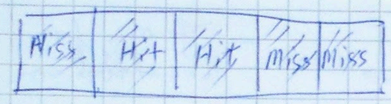
Inertial Sensors

* In Self Driving Car

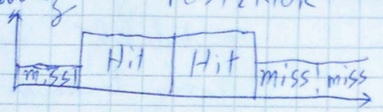
  - Robot Model (bicycle model)
  - Sensor data
  - gps, inertials ...

→ Localization





probability



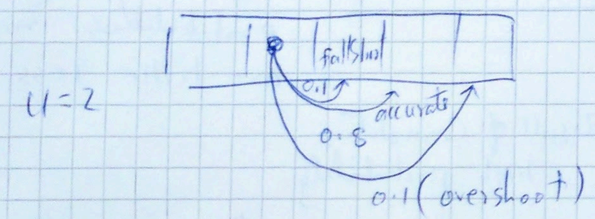
normalized & localized

location

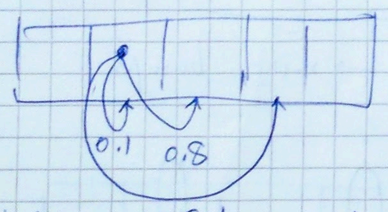
* Algo for shifting array in both direction.

```
def move(move):
    q = []
    for index in range(len(oldArray)):
        q.append(p[(index - mov) % len(oldArray)])
    return q
```

* Inaccurate Robot Motions:



* Undershoot
* Exact
* Overshoot



* So a robot a probability that it won't move correctly to the desired location, having 20% chances of error. It makes localization hard. B/c we predict based on how much robot moved & if this movement is inaccurate - then prediction is inaccurate

(1) - So Accurate movement ; Inaccurate movement

U = 2

$P(x_{i+2}) = 1$

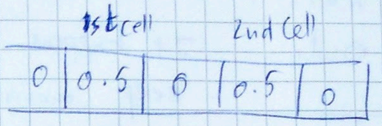
can't move here: $P(x_{i+1}) = 0$, $P(x_{i+3}) = 0$

U = 2

$P(x_{i+2}) = 0.8$

$P(x_{i+1}) = 0.1$, $P(x_{i+3}) = 0.1$ } Can move here by Error

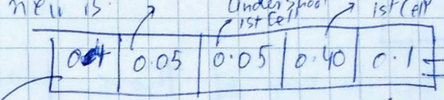
(2)



U = 2

$P(x_{i+2} \mid x_i) = 0.8$, $P(x_{i+1} \mid x_i) = 0.1$, $P(x_{i+3} \mid x_i) = 0.1$

So new is:

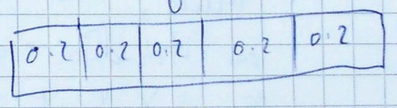


Notice, robot is very uncertain about its movement or location after movement

Exact of 2nd cell

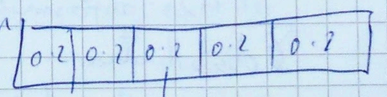
{ Pexact, Povershoot, Pundershoot }

(3) In case of Uniform Distribution



$P(x_{i+2} \mid x_i) = 0.8$; $P(x_{i+1} \mid x_i) = 0.1$; $P(x_{i+3} \mid x_i) = 0.1$

Then



$(0.2 \times 0.1) + (0.2 \times 0.8) + (0.1 \times 0.1)$ → Convolution (add the products)

which means, there is no more uncertainty than uniform distribution

---

* Algo for adjusting the values

```
value_exact = p[selected + 0] * pExact
value_undershoot = p[selected + 1] * pUndershoot
value_overshoot = p[selected - 1] * pOvershoot
Σ(value_exact + ... overshoot + .. undershoot)
```

⇒ Limit Distribution

Initially we know where we're

1	0	0	0	0

$$\begin{bmatrix} U = 2 \\ \text{means:} \\ \text{exact} = 0.8 \\ \text{under} = 0.1 \\ \text{over} = 0.1 \end{bmatrix}$$

If a robot keeps moving with inaccuracy in its movements not corrected, in the end, it'll be on uniform distribution, called "stat of maximal uncertainty".

It happens when we keep moving w/o sense step

now we don't where one's robot

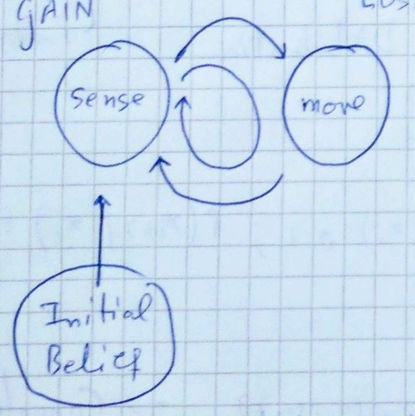
0.2	0.2	0.2	0.2	0.2

thinks he could be in any of those places.

$$p(x_4) = \begin{cases} 0.8\,p(x_2) \rightarrow \text{exact} \\ 0.1\,p(x_1) \rightarrow \text{undershoot} \\ 0.1\,p(x_3) \rightarrow \text{overshoot} \end{cases} \quad \text{in case of } U=2$$

⇒ Localization:

GAIN Lose



⇒ Entropy (Measure of info a distribution has)

$$-\sum p(x_i) \log p(x_i)$$

update (move) step, entropy ↓

measurement (sense) step, entropy ↑

- We provide initial belief, then its an iteration of "sense" & "motion".
- After every move, it loses information & distribution is flatter than before.
- After every sense, it gains information & distribution is more focused than before

* Summary of Localization:

- Belief = probability
- sense = product (w/ normalization)
- move = convolution (addition)

after each motion, we guess where the robot might have come from

* Bayes's Rule is most fundamental for probabilistic inference

$x \rightarrow$ grid cell index ; $z \rightarrow$ measurement

$P(x_i \mid z)$ = normalized posterior distribution

$$= \frac{P(z \mid x_i)\,P(x_i)}{P(z)}$$

$P(z \mid x_i)$: measurement probability → prior distribution

$P(z)$ → normalization factor

chances of seeing red/green tile for every possible location

this makes measurement probability devoid of any location information

$$\bar{P}(z) = \sum P(z \mid x_i) \cdot P(x_i)$$

So:

- $\bar{P}(x_i \mid z) \leftarrow P(z \mid x_i)\,P(x_i)$
  - non-normalized.
- $\alpha = \sum \bar{P}(x_i \mid z)$ → sum of all non-normalized
- $P(x_i \mid z) = \frac{1}{\alpha}\,\bar{P}(x_i \mid z)$

$$P(x_i \mid z) = \frac{1}{\sum \bar{P}(x_i \mid z)}\,P(z \mid x_i)\,P(x_i)$$

- normalized posterior distribution
- normalizer
- measurement probability
- prior distrib.

* General Baye's Rule:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

where $P(B) = P(B|A) * P(A) + P(B|>A) + P(>A)$

→ Question:

$P(c) = 0.001$; $P(>c) = 0.999$

$P(Pos|c) = 0.8$; $P(Pos|>c) = 0.1$

find

$P(c|Pos) = ?$

Sol

$\bar{P}(c|Pos) = P(Pos|c) * P(c)$.

$= (0.8)(0.001) = 0.0008$

And:

$\bar{P}(>c|Pos) = P(Pos|>c) * P(>c)$

$= (0.1)(0.999) = 0.0999$

So:

$$P(c|Pos) = \frac{\bar{P}(c|Pos)}{\bar{P}(c|Pos) + \bar{P}(>c|Pos)} = \frac{\bar{P}(c|Pos)}{\sum P(Pos)}$$

$$P(c|Pos) = \frac{0.0008}{0.0008 + 0.0999} = 0.0079$$

→ probability, localization, Baye's Rule

* Localization Program

formulas for Baye's Rule:

* $P(A|B) = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(B|>A) * P(>A)}$

* $P(B|A) = \frac{P(A|B) * P(B)}{P(A|B) * P(B) + P(A|>B) * P(>B)}$

⇒ Algorithm for sense:

```
def sense(probabilities, measurement):
    q = []
    * for index in range(len(probabilities)):
            if (measurement is world[index]):
                q.append(p * pHit)
            else:
                q.append(p * pMiss)
    * s = sum(q)
      for(i in range(len(q)):
          q[i] = sum
      return q
```

⇒ Algorithm for move

```
def move(probabilities, how_much):
    * q = probabilities[:]
    * for index in range(len(probabilities)):
          q[how_much] = probabilities[index]
          how_much = (how_much + 1) % len(p)
    * return q
```

⇒ Algorithm for move w/ inaccuracies

```
def move_inaccurate(probabilities, how_much):
    * q = p[:]  # OR q = [None] * len(probabilities)
    * for index in range(len(probabilities)):
          q[how_much] = (probabilities[index] * pExact) +
                        (probabilities[(index-1) % len(q)]) * pOvershoot
                      + (probabilities[(index+1) % len(q)]) * pUndershoot
          how_much = (how_much + 1) % len(q)
```

→ Algorithm for localization : (Sense & move).

```
p = [0.1, 0.2, 0.2, 0.1, 0.4] ; pHit = 0.8 ; pUnder = 0.2
world = ['r', 'g', 'r', 'g', 'r'] ; pMiss = 0.2 ; pExact = 0.7
measurements = ['r', 'g', 'g']   # me        ; pOver = 0.1
moves = [1, 2, 1]                # mo
```

```
def perform_localization(p, me, mo):
    for index_0 in range(Len(me)):
        q = []
        for index_1 in range(Len(p)):
            if (me[index_0] is world[index_1]):
                q.append(p * = pHit)
            else:
                q.append(p * = pMiss)
    → p = q[:]
    for (index_2 in range(Len(p))):
        p[moves[index]] = q[index_2] * pExact +
                          q[(index_2 + 1) % len(p)] * pUnder +
                          q[(index_2 - 1) % len(p)] * pOver
        index_2 = (index_2 + 1) % Len(p)
    print p
```

→ Summary :

1)- for Sense : [pHit, pMiss, world, measurements]

2)- for move : [pExact, pUnder, pOver, movements]

→ Bayes Rule

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(B|\sim A) * P(\sim A)}$$

→ Total Probability Theorem :

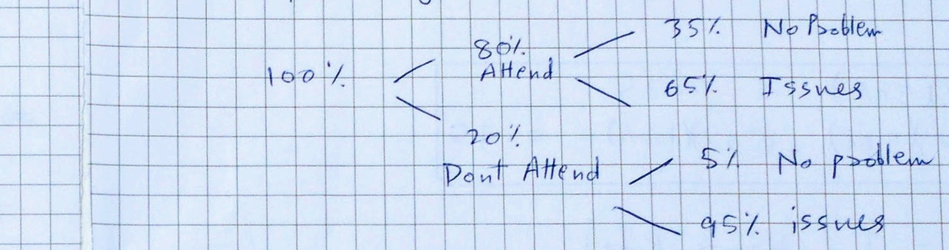
$$P(A) = \sum_{i.e} [P(A|B) * P(B)] \longrightarrow \text{Convolution (Sum of product)}$$

$P(A|B)P(B) + P(A|\sim B)P(\sim B)$

where : B → possible prior locations

Total Probability

Example - ~~Bayes~~ Theorem



Find probability that a random person has no problems :

P(no problem) = (35%)(80%) + (5%)(20%)

P(No prob) = 29%

* Spam Filters in mail are implemented using Naive Bayes Rule :

$$P(Spam|Word)' = \frac{^2P(Word|spam)\ ^3P(spam)}{P(Word|spam)P(spam) + P(Word|\sim spam)P(\sim spam)}$$

1 - probability a msg is spam given that[4] a word[5] is there.

2 - The word is there ~~given~~ in spam.

3 - A msg is spam

4 - The word doesn't appear in spam

5 - A msg is not spam

⇒ Example - 2:

Fair Coin; T = 50% , H = 50%

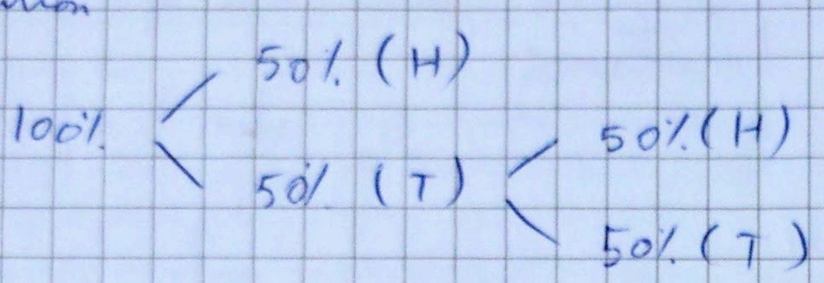
when:

HT → accept

TH → flip; accept

find P(T).

Solution



Sol:

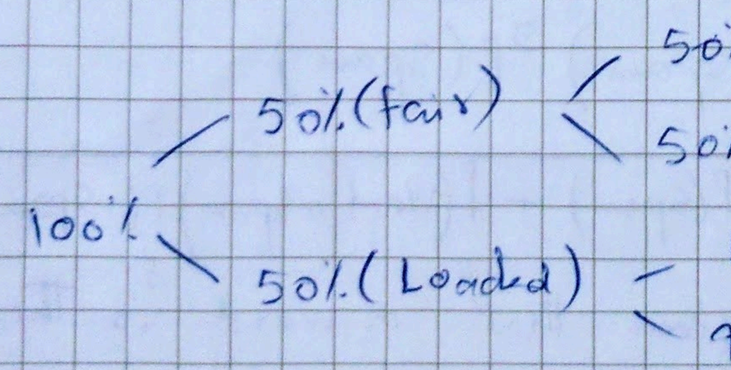
$$P(T) = (50\%)(50\%) = 0.125$$
$$P(H) = (50\%)(50\%) + (50\%)(100\%) = 0.75$$

⇒ Example - 3 [Bayes Rule]:

Consider Loaded Coin, 50% chance that coin is fair & 50% that its Loaded. find probability of fair given its head.

Sol.

$$P(\text{fair}|H) = \frac{P(H|\text{fair})\,P(\text{fair})}{P(H|\text{fair})P(\text{fair}) + P(H|\sim\text{fair})\,P(\sim\text{fair})}$$



$$P(\text{fair}|H) = \frac{(50\%)(50\%)}{(50\%)(50\%) + (10\%)(50\%)}$$

$$P(\text{fair}|H) = 83\%$$