AI for Robotics

* return q

1 Skimming Kalman filters — Sebastian Thrun

LASER Range finder, gps

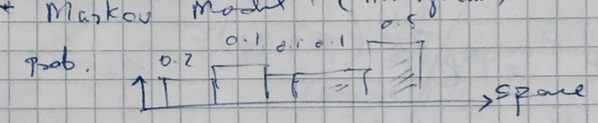
used for tracking, similar to Monte Carlo localization

* Kalman
  - Continuous
  - Uni Modal
  - Guassian Distribution

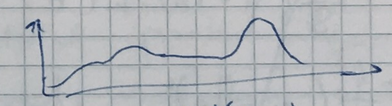
Monte Carlo
- Discrete
- Multi Modal
- Normal Distribution
- Histogram

Particle filter
- Continuous
- Multi Modal

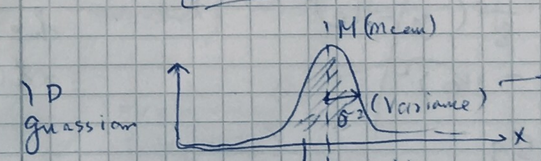
* Markov Model. (Histogram)



So:



→ Normal Distrib. (Monte Carlo)



→ Kalman filter (Guassian)

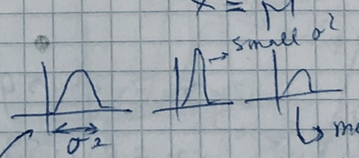
1) Continuous
→ area is 1 (represents probability)

So, rather than histogram we use guassian distribution (μ, σ^2) for estimation

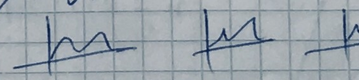
$$f(x) = \exp^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right); \quad x \rightarrow \text{Random Variable}$$

↳ for normalization

when $x = \mu$



→ guassians
* Single Peak
* Expo
* UniModal

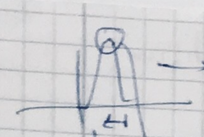


→ * Bimodal

large σ^2

So: Larger σ^2 means $f(x)$ matters lesser
So lesser peak & more uncertain in System

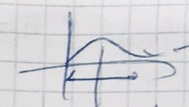
* return q



→ high Confidence

↳ b/c high prob.

↳ Low σ^2



→ low Confidence

↳ low probability

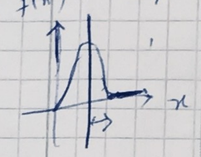
↳ High σ^2

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

* Gaussian is used as "belief" in Kalman

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

(P) $f(x)$ 0.1209



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

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

$\log(e^8)$

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

$8\log(e)$

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

$\ln e^x = \ln 55$

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

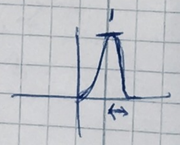
$$12.896 = (x-10)^2$$

$$x - 10 = 3.59$$

$$\boxed{x = 13.59}$$

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

$x =$



0.199

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

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

$x = 24.59$

$x = -4.59$

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

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

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

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

$$\boxed{f(x) = 1 \Rightarrow x = \mu \;\&\; \sigma^2 = \frac{1}{2\pi}}$$

* Kalman filter

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

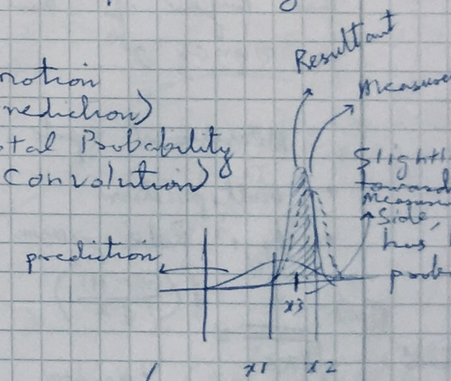
motion (predict) ↳ requires convolution

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

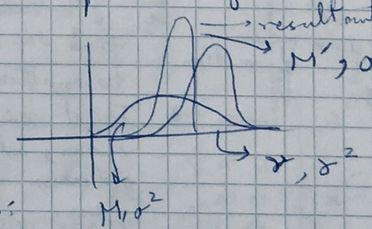
in motion, we keep track of where total probability went when we moved, using Law of total probability.

Measurement (Update) Bayes Rule (Product)

Motion (prediction) Total Probability (Convolution)



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



* And the resultant probability will be higher than both prediction & update

↳ σ' have lesser Covariance

So:

$$\Rightarrow \mu' = \frac{\mu(r^2) + \nu(\sigma^2)}{r^2 + \sigma^2} \rightarrow \text{weighted Sum}$$

$$\Rightarrow \sigma'^2 = 1/(1/r^2 + 1/\sigma^2) \rightarrow \text{Unaffected by means}$$

} update resulting

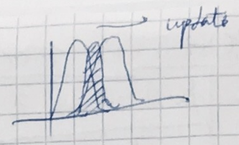
* return q

~~Ex~~ So 1

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

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

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



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

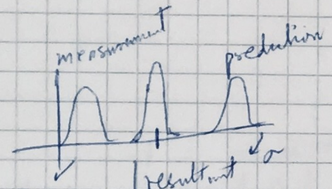
$\nu = 13 \; ; \; r^2 = 2$

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

b/w μ & ν

→ Less than σ^2 & r^2 which means high probability

→ on each update, probability increases



→ high probability, low variance

Sense (Measurement):

$\mu' = \mu_1\sigma_1^2 + \mu_2\sigma_2^2 / \sigma_1^2 + \sigma_2^2$

$\sigma'^2 = 1/(1/\sigma_1^2 + 1/\sigma_2^2)$

Result

move:

$\mu' = \mu_1 + \mu_2$

$\sigma'^2 = \sigma_1^2 + \sigma_1^2$

As:

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

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

Measurement Update

* Multiplication
* Bayes' Rule

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

probability

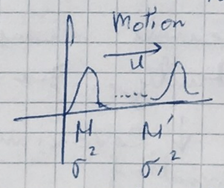
mean

Random Value

Variance (Confidence)

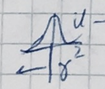
Motion Update

* 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)(ii)

Kalman Filter In Higher Dimension

(3)

⇒ Prediction Step

$x' \rightarrow$ new estimate

$x \rightarrow$ estimate

$F \rightarrow$ State transition matrix

$u \rightarrow$ motion vector

$P \rightarrow$ Uncertainty Covariance

move

new estimate

$$\boxed{\begin{aligned} x' &= Fx + u \\ P' &= FPF^T \end{aligned}}$$

⇒ Measurement Step:

$z \rightarrow$ measurements

$H \rightarrow$ measurement function

$I \rightarrow$ Identity Matrix

$R \rightarrow$ Measurement noise

$y \rightarrow$ new measurement

Kalman Gain

sense

new measurement

Projecting System Uncertainty into measurement Space

$y = z^t - H \cdot x$

↳ should be in matrix form

$S = H \cdot P \cdot H^T + R$

↳ error

measurement noise

$\leftarrow K = PH^T \cdot S^{-1}$

$x' = x + (Ky) \rightarrow$ Updated Estimate

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

Steps:

sense

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:

$$\boxed{x' = x + K \cdot y}$$

Find new uncertainty:

$P' = (I - KH) \cdot P$

move

Step - 2:

$$\boxed{P' = FPF^T}$$

$x = Fx + u$

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

$\sigma' = \frac{\sigma^2}{2}$

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

$(1 \quad 0)$

$x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; P = \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$ → 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

$R = [1]$

$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$z = [1, 2, 3]$

matrix([[m[n]]])

$x = Fx + bu$

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

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

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

$x = Fx + u$

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

2

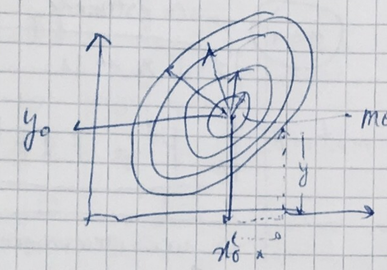
3 [2.998

2 [0.999]

* return q

③ - Multivariate Gaussians for Kalman filter

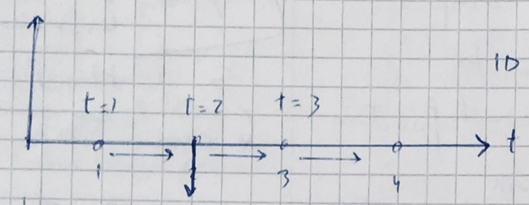
(i)



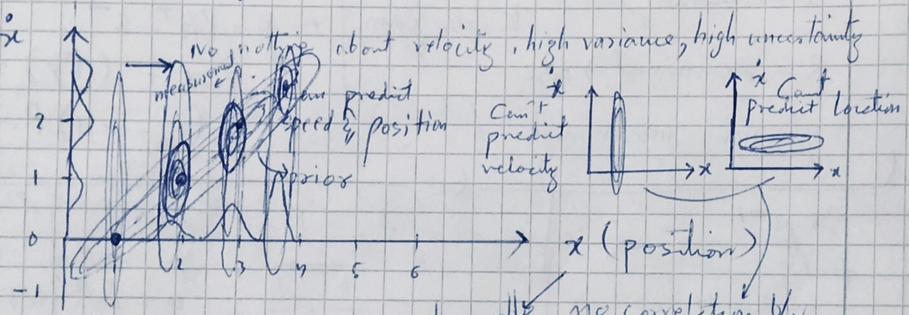
2D Guassian

- mean is repres. by Contours.

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



Inferable (velocity) $\dot{x}$



observable no correlation b/w x & $\dot{x}$

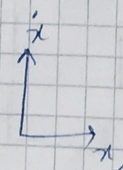
$$x' = x + \dot{x}t$$

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

- Kalman filter States

Observables

Hidden



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

F → Δt

x' = new pose; $\dot{x}'$ = new velocity

$x' = x + \dot{x}(1)$

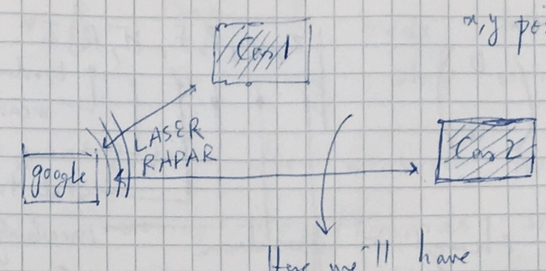
$\dot{x}' = \dot{x}$

uniform speed.

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

z → measurement; H → only position; no velocity (velocity not observable)

④



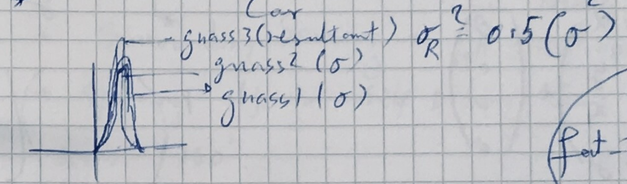
x, y positions

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

x, y velocities

Here we'll have correspondent problem where we don't know the data is from which Car

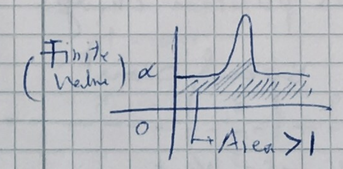
*



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

Heavy tail Distr.
- fat-tail
- Long tail
- Subexponential

* Heavy tail Guassian



Its not possible to represent tail guassian with:

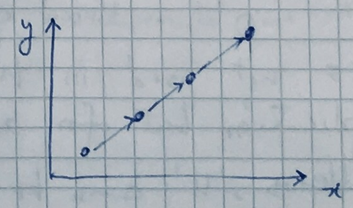
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)}$$

B/c

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

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

- $\begin{pmatrix} x \\ \dot{x} \end{pmatrix}$ → 2 dimensional — x dim. pose, x dim. velocity



(see previous diagram like this, but that was for x & $\dot{x}$)

- $\begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix}$ → 4 dimensional

* return q

for 1D:

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

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

- x → estimate matrix
- u → motion vector
- P → covariance matrix
- F → state transition matrix
- H → measurement matrix
- R → measurement noise
- I → identity matrix

→ Kalman filter assumes constant velocity

So that:

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

$\dot{x}' = \dot{x}$ → const. speed.

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 filter 4 dimensional — 4 state vectors

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

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

$y = \frac{5}{t}$

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

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

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

5

- R → represents noise in the sensor. In case of white noise It is 0
- Radar also measures how fast things are moving away/towards using doppler effect.
- Lasar (Lidar) has higher spatial resolution
- Bayes Rule is used to combined results of both.
- Surrounding Sensing → then RADAR can look in all directions.
- Spinning obj in self driving car → like, it scans 360°.

- 1D Kalman filter:

$dt = 0.1$

$x = \begin{pmatrix} -5 \\ 5 \\ 0 \\ 0 \end{pmatrix}$; $P = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 1000 \end{pmatrix}$; $u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

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

$z = [(1,4), (6,0), (11,4), (16,5)]$

for i in range

$z = \text{np.matrix}([z[\text{index}]])$ → transform lists element to matrix

- Answer → Adaptive with Gaussian Noise as noise components are uncorrelated.

- return q

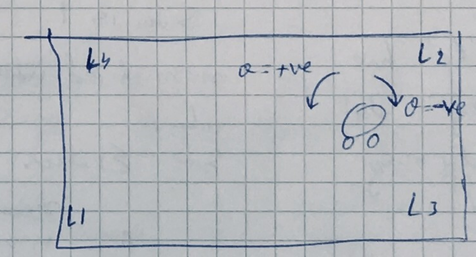
⇒ Particle filters.

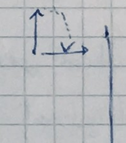
- Histogram filter scales exponential while Kalman filter scales quadratically. So kalman filter is more efficient.

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

- Particle
  - → Continuous, Multimodal, Approx.
  - → shouldn't be used for ~~any~~ more than 4 dimensions.

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





rotation direction

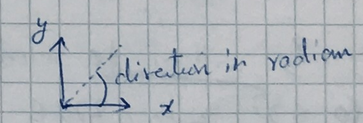
set(x, y, θ)
move(dyn, ...)
sense

30, 50

for : 5.0
turn : 0.1
sense : 5.0

⇒ N = 1000 → (x, y, direction)

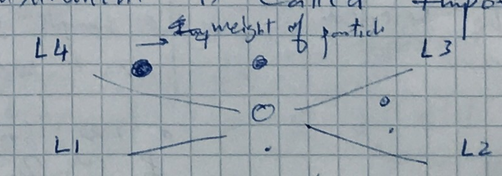
↳ each of the dots is 3x1 vector



turn by 0.1
move by 5

⇒ mismatch b/w predicted & actual...

... measurement is called Importance Weight



Higher weight particles survive

* return q

⇒ Selecting particles w.r.t to their weights so high probability particles remain.
↳ Resampling

In the end only

* transational noise
* rotational noise
* measurement noise

robot.sense( )

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

⇒ Resampling

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

normalized_value = $\frac{\text{Value}^{1\ldots 2}}{\text{Sum}}$

where
Sum = Σ values

	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 –	$= 1.2$,	$\alpha_5 = 0.2$
	$\Sigma = 6$	$\Sigma = 1$

Here, even though 'P_3' 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.

* Prob. that of

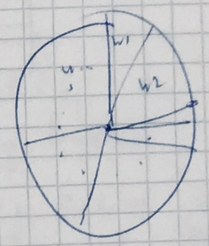
* Program to take lists of particle & weights & rearrange or sample particle w.r.t weights.

* Does not guarantee the w.r.t sorted w. The rearrange p

⇒ Resampling wheel Algorithm:

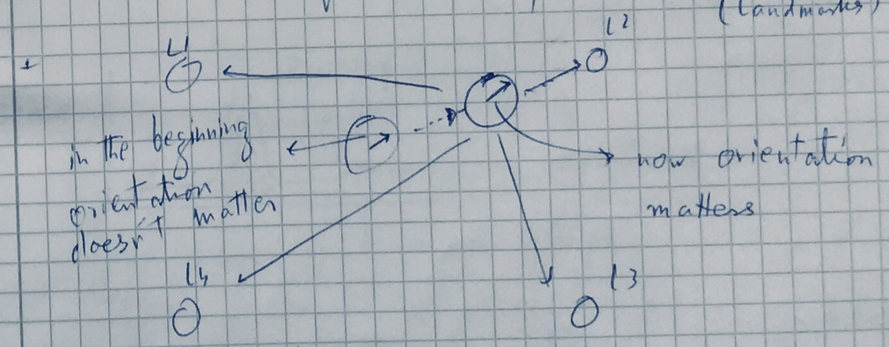
* It reorder the ~~value~~ particles w.r.t their probabilities.

Algorithm:
w → weights of particle (array)
index = random(0 — N) [int]
beta → 0 — # of iterations
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 landmarks (landmarks)



* So orientations

e.g. (x, y)

⇒ Measurement update:

$P(x|z) \propto P(z|x)\ P(x)$

correct posterior ← resampling ↑ importance weights; $P(x)$ → distribution ↳ particles

⇒ Motion Update:

correct distribution after robot motion new position ← $P(x') = \sum P(x'|x)\ P(x)$

sample; $P(x)$ → particles; $P(x'|x)$ → motion model

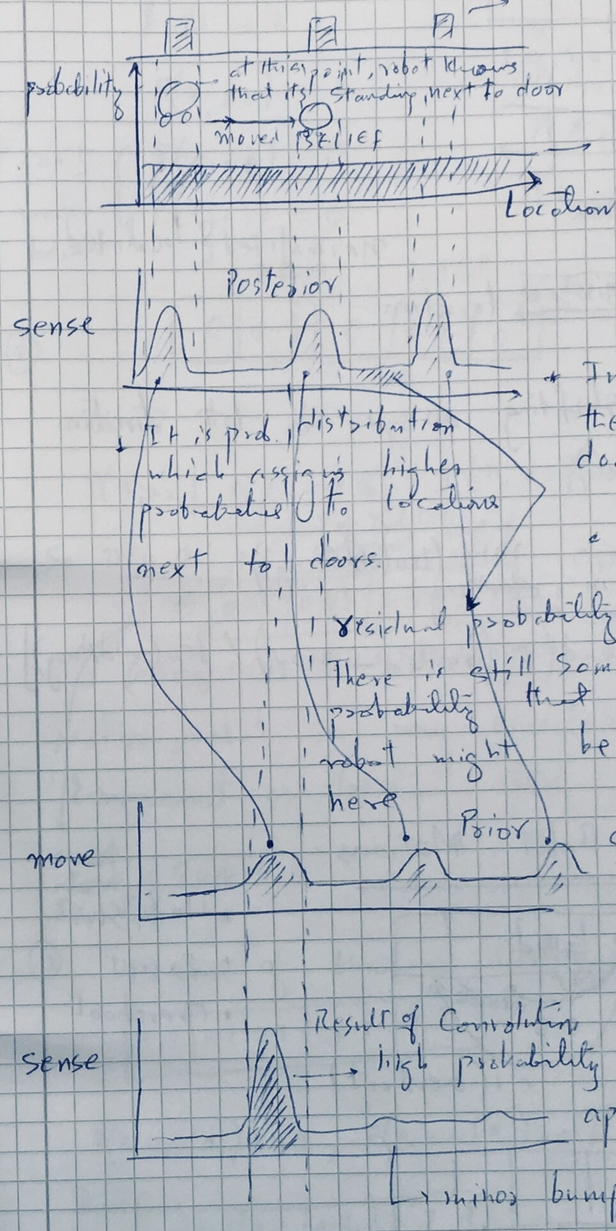
* return q

Inertial Sensors

* In Self Driving Car

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

→ Localization



→ refuses for localization

Uniform Maximum Confusion => Initially we assign same probability to every location

* Increased Belief in the locations of the doors

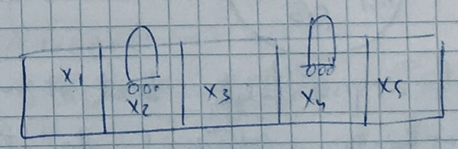
* Posterior → Belief A/f Measurement

* It is prob. distribution which assigns higher probabilities to locations next to doors.

Residual probability ⇒ There is still some probability that robot might be here

Convolution ; robot moved forward probabilities now are little less

$\frac{1}{5}$



probability of each of these is 0.2 $\left[\frac{1}{5}\right]$

* this is not a correct distrib. as it doesn't sum up to 1. We can normalize it

$P(x_i \mid z)$ → measurement

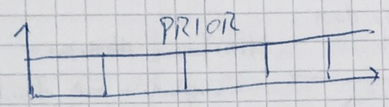
x_i → any value

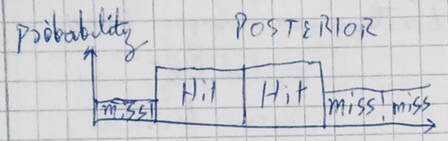
posterior distribution

→ probability of localizing after measurement.

* return q

Miss	Hit	Hit	Miss	Miss
0.2	0.2	0.2	0.2	0.2



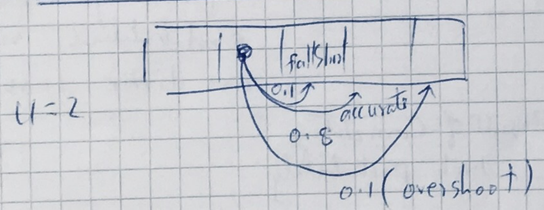


normalized & localized

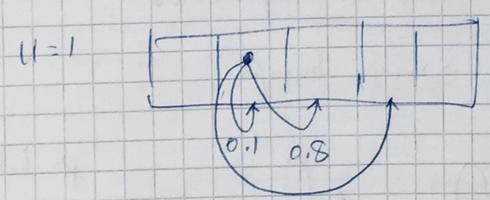
* Algo for shifting array in both direction

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

* <u>Inaccurate Robot Motions:</u>



* Under Shoot
* Exact
* Overshoot



* So robot a probability that it won't move correctly to the desired location, having 20% chances of error. Its 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 $\begin{cases} P(x_{i+1}) = 0 \\ P(x_{i+3}) = 0 \end{cases}$

$u = 2$

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

$P(x_{i+1}) = 0.1$

$P(x_{i+3}) = 0.1$

} Can move here by Error

(2)

	1st cell		2nd cell	
0	0.5	0	0.5	0

$u = 2$

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

So new is:

0	0.05	0.05	0.40	0.1
(0.5×0.8)	0.5×0.1	0.5×0.1	0.5×0.8	$2(0.5\times0.1)$

overshoot of 2nd cell; Undershoot 1st cell; Shoot of 1st cell

→ overshoot of 1st cell

→ undershoot of 2nd cell

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

~~overshoot~~ Shoot of 2nd cell

Exact

$\left\{ \begin{matrix} P_{exact} \\ P_{overshoot} \\ P_{undershoot} \end{matrix} \right\}$

(3) In case of Uniform Distribution

0.2	0.2	0.2	0.2	0.2

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

Then

0.2	0.2	0.2	0.2	0.2

$(0.2\times0.1) + (0.2\times0.8) + (0.2\times0.1)$ → Convolution (add the products)

which means, ~~the case of~~ there is no ~~more~~ uncertainty 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)
(how much + 1) % len(p)
```

* return q

→ Limit Distribution

Initially we know where we're

1	0	0	0	0

$U = 2$

means:
exact = 0.8
under = 0.1
over = 0.1

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

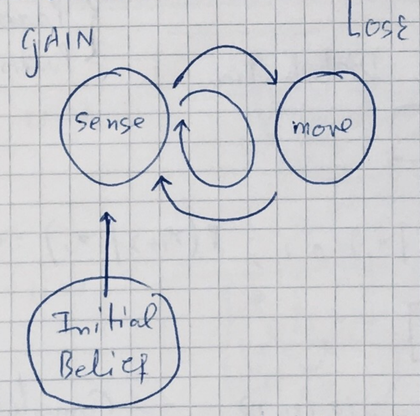
It happens when we keep moving w/o sense step

now we don't know where the robot thinks he could be in any of those places.

0.2	0.2	0.2	0.2	0.2

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

→ Localization:



- 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

→ Entropy (Measure of info a distribution has)

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

update (move) step, entropy ↓

measurement (sense) step, entropy ↑

* Summary of Localization:

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

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

* Baye's Rule is most fundamental for probabilistic inference → sense

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

$$\underbrace{P(x_i \mid z)}_{\text{normalized posterior distribution}} = \frac{\overbrace{P(z \mid x_i)}^{\text{measurement probability}}\ \overbrace{P(x_i)}^{\text{prior distribution}}}{\underbrace{P(z)}_{\text{normalization factor}}}$$

Chances of seeing red/green tile for every possible location

→ this make measurement probability devoid of any location information

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

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

$$\boxed{P(x_i \mid z) = \underbrace{\frac{1}{\sum \bar{P}(x_i \mid z)}}_{\text{normalizer}}\ \underbrace{P(z \mid x_i)}_{\text{measurement probability}}\ \underbrace{P(x_i)}_{\text{prior distrib.}}}$$

normalized posterior distribution

* return q

* General Bayes'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:

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

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

* probability, localization, Bayes's Rule
* Localization Problem

formulas for Bayes's Rule:

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

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

⇒ 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(q)
    * 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)
    * return q
```

* General Baye's Rule:

→ 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]

⇒ Baye's 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:

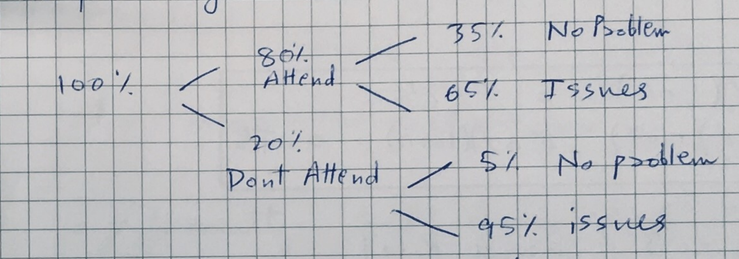
$$P(A) = \sum_i \left[P(A|B) * P(B) \right] \longrightarrow \text{Convolution (Sum of product)}$$

where: i.e $P(A|B)P(B) + P(A|\sim B)P(\sim B)$

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%)

$$\boxed{P(\text{No prob}) = 29\%}$$

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

$$P(\text{Spam} | \text{Word})^1 = \frac{{}^2P(\text{Word}|\text{Spam})\,{}^3P(\text{Spam})}{P(\text{Word}|\text{Spam})P(\text{Spam}) + P(\text{Word}|\sim\text{Spam})P(\sim\text{Spam})}$$

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

* General Baye's Rule:

⇒ Example - 2:

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

when:

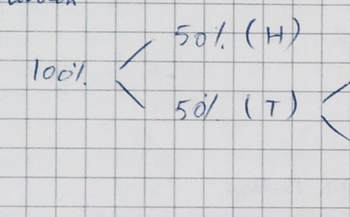
HT → accept

TH → flip ; accept

find P(T).

Solution

P(Spam | word) →	msg is Spam & containing word.
P(word | Spam) →	word appears in Spam msg.
P(word | ~Spam) →	word appears in non-spam msg.
P(word) →	word appears in any msg.
P(~Word) →	word doesn't appear in any msg



So:

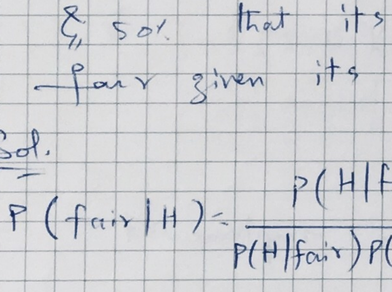
$$\boxed{\begin{aligned} P(T) &= (50\%)(50\%) = 0.25 \\ P(H) &= (50\%)(50\%) + (50\%)(100\%) = 0.75 \end{aligned}}$$

⇒ 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.

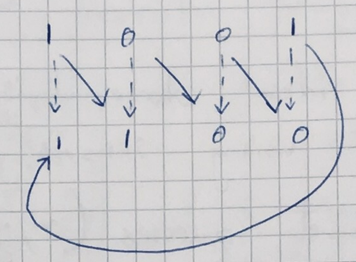
$$P(\text{fair} \mid H) = \frac{P(H \mid \text{fair})\, P(\text{fair})}{P(H \mid \text{fair})\, P(\text{fair}) + P(H \mid \sim\text{fair})\, P(\sim\text{fair})}$$



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

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

* Diagram for move:



↓ → moved

⋮ → not moved

* 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 simultaneous w/ Localization

* Currently self drive doesn't work in snow.

* electro steering booster → control steering using motor

* LEGO minestorm.

* Stanley had 4 embeded controllers & could only see forward.

* Junior had 2 quad core CPUs, LIDAR & Radar.

* LIDAR spins 360° to form map of the environment.

* Sensor fusion b/w LIDAR & Radar is done using Bayes Rule i.e product of probabilities.

* for range sensing, SONAR is used.

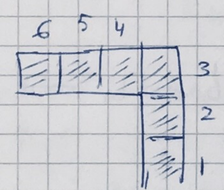
* global Localization:
  * Robot has no idea where it is
  * we localize it using sensors data
  * particles → discrete guesses (x, y, θ)

    heading direction

⟹ Search for Robot Motion:

* Planning → process of finding path from point A → B.

* Cost



* In real traffic, left turn has high cost b/c Left ~~hand~~ turn are harder to do than right turn.

* Fedex & UPS avoids path w/ Left turns.

* Breadth first Search.

* A* Heuristic Search

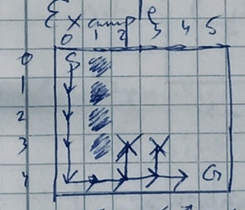
10	9	8	7	6	5
9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1
5	4	3	2	1	0

* $h(x, y) \leq$ dist. to goal.
* Manhattan distance.
* Manhattan heuristic could be used in freeform roaming
* In reality, we used euclidean distance to a target location
* If $h(x,y) = 0$ Then it is same as Breadth first search.

* elements are sorted w.r.t f now:

$$\boxed{f = h(x, y) + g}$$

Example



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

format
* open [x, y], g, f → g+h
* //// → taken

open ~~[0, 0], 0, 9~~

~~[1, 0], 1, 9~~

~~[2, 0], 2, 9~~

~~[3, 0], 4, 9~~

~~[4, 0], 5, 9~~

~~[4, 1], 6, 9~~

~~[4, 2], 7, 9~~

[3, 2], 8, 12

~~[4, 3], 8, 11~~

[3, 3], 9, 12

~~[4, 4], 9, 11~~

[4, 5], 9, 9

Dynamic Programming :

* In real world, a car loses track of its starting position, in that case dyn. prog. is very helpful.
* dyn. progr. gives optimal action at every single grid cell.
* every cell has a Label ⟶ policy.
* Stochastic environment :
  outputs of any action are non-deterministic.
* policy :
  A fn. which maps grid cell to action
  $(x, y) \rightarrow$ action $(\uparrow \leftarrow \downarrow \rightarrow)$

$$\boxed{\min_{x,y} f(x', y') + \text{cost} \rightarrow f(x, y)}$$

optimal neighbours

1 in our case

value of optimal neighbours

* Value fn. is obtained by applying above equation recursively.
* Once we've the value function, we find optimal control action ~~which~~ by minimizing value, which is a Hill Climbing type of action

10	9	8	7	6	5	4
11			8	7		3
12	13		9	8		2
13	14		10	9		1
14	13	12	11		1	0

→ dynamic policy

12	11		7	6	5
11	10		6	5	4
10	9		5	4	3
9	8	7	6		2
10	9				1
11	10	11	12		0

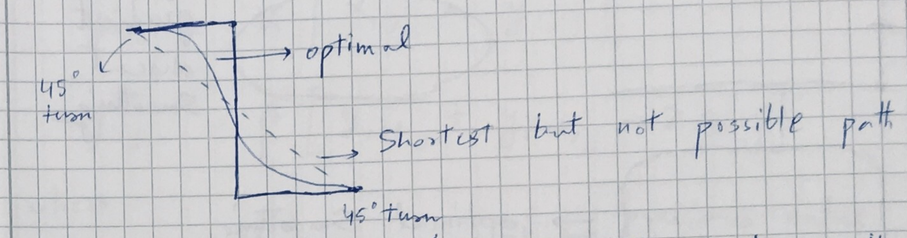
---

* Self driving don't record/process audio
* A* assumes static & deterministic world.
* A* doesn't work with branching output e.g flip coins, 2 nodes having same heuristic $h(x, y)$.
* A* doesn't work w/ information gathering which is needed in planning.
* Dynamic programming is computing intensive & doesn't scale well for large state space.
* In ~~A*~~ Stanley, A* was used which was replanning very quickly & very frequently to overcome problem of changing world
* In real world, planning is done within visual horizon $\boxed{(x, y, \theta, \text{velocity})}$

A* & dynamic programming allows multiple goals in a path.

* Real world heuristic → we do planning in 2D (x, y) & used this planning as heuristic for 3D (x, y, θ) planning.

⇒ PID Control



* Smoothing Algorithm (Gradient Descent Method):

1 → $(y_i - x_i)^2$ → min

2 → $(y_i - y_{i+1})^2$ → min

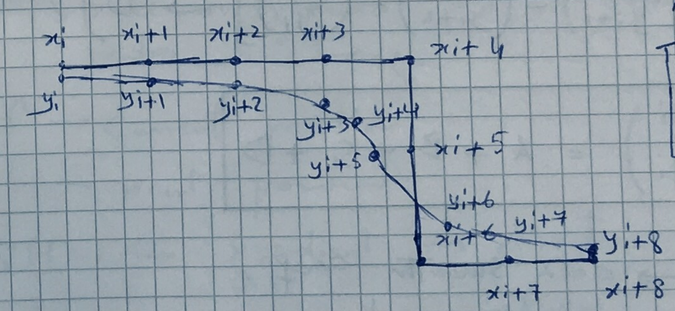
So:

$(y_i - x_i)^2$ → min. error from original (x_i) to Smooth (y_i) point.

$(y_i - y_{i+1})^2$ → minimize dist. b/w consecutive point.

when

* $x_i = y_i$ → original path.
* $y_i = y_{i+1}$ → a point

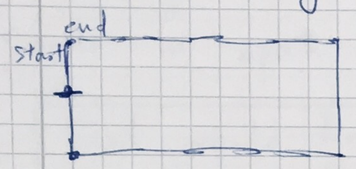


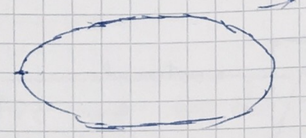
Always

$$\boxed{\begin{aligned} x_0 &= y_0 \\ x_{n-1} &= y_{n-1} \end{aligned}}$$

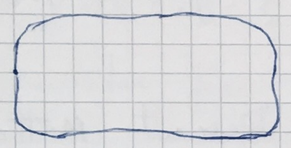
$$\boxed{\begin{aligned} y_i &= y_i + \alpha(x_i - y_i) \\ y_i &= y_i + \beta(y_{i+1} + y_{i-1} - 2y_i) \end{aligned}}$$

* Constrained Cyclic Smoothing





→ Cyclic gradient descent smoothing



→ optimal smoothing

So

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} \rightarrow \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \gamma \left[2\begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} - \begin{pmatrix} x_{i-2} \\ y_{i-2} \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right]$$

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} \rightarrow \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \gamma \left[2\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} - \begin{pmatrix} x_{i+2} \\ y_{i+2} \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix} \right]$$

* So

* $\alpha(x_i - y_i) \rightarrow A$
* $\beta(y_{i+1} - y_{i-1} - 2y_i) \rightarrow B$
* $\gamma\left[2\begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} - \begin{pmatrix} x_{i-2} \\ y_{i-2} \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix}\right] \rightarrow C$
* $\gamma\left[2\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} - \begin{pmatrix} x_{i+2} \\ y_{i+2} \end{pmatrix} - \begin{pmatrix} x_i \\ y_i \end{pmatrix}\right] \rightarrow D$

} Smoothing

&

$$\boxed{\begin{pmatrix} x_i \\ y_i \end{pmatrix} = A + B + C + D}$$

where

$\alpha \rightarrow$ data smooth coeff.

$\beta \rightarrow$ weight smooth coeff.

$\gamma \rightarrow$ constrained weight smooth coeff.

&

$\gamma = \frac{1}{2}(\beta)$ usually.

Maths behind Gradient Descent Method:

$$y_{i,j} = \alpha\left[x_{i,j} - y_{i,j}\right]$$

$$y_{i,j} = \beta\left[y_{i+1,j} + y_{i-1,j} - 2y_{i,j}\right]$$

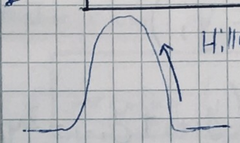
where

$x \rightarrow$ old x, y ; $i-1 \rightarrow$ previous value

$y \rightarrow$ new x, y ; $i+1 \rightarrow$ next value

$i \rightarrow$ current value

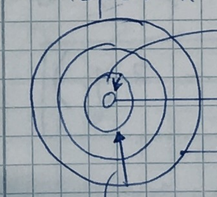
* $\boxed{b = a - \gamma \nabla F(a)}$ → minimization i.e subtract the gradient



Hill climbing

* When hill is steep, we can take big step.
* When hill is not steep, step size should be smaller.
* This is so that we avoid overshooting the target.

* Topdown View



→ small vector as near to top

→ top level of Hill

→ ground level of Hill

Since on ground, Hill is steep so vector is long

$a \rightarrow$ Current position

$b \rightarrow$ next position

$\nabla f(a) \rightarrow$ gradient → direction of steepest ascent

$-\nabla f(a) \rightarrow$ direction of steepest descent

$$f(y_i) = \frac{\alpha}{2}(x_i - y_i)^2 + \frac{\beta}{2}(y_i - y_{i+1})^2 + \frac{\beta}{2}(y_i - y_{i-1})^2$$

↳ minimize with gradient descent by iterating over $b = a - \gamma(\nabla f(a))$ until we get shallow slope. i.e on top of hill

So gradient here will $f'(y_i)$

Derivation

Consider the equations:

$$y_i' = \alpha(x_i - y_i)$$

$$y_i = \beta(y_{i+1} + y_{i-1} - 2y_i)$$

i.e

$$y_i' = y_i + \alpha(x_i - y_i) + \beta(y_{i+1} + y_{i-1} - 2y_i)$$

For derivation, consider:

$$f(y_i) = \frac{\alpha}{2}(x_i - y_i)^2 + \frac{\beta}{2}(y_i - y_{i+1})^2 + \frac{\beta}{2}(y_i - y_{i-1})^2$$

$$\boxed{y_i' = y_i + \alpha(x_i - y_i) + \beta(y_{i+1} + y_{i-1} - 2y_i)}$$

* For smoothing, <u>Linear Regression</u> method is also used.

where:

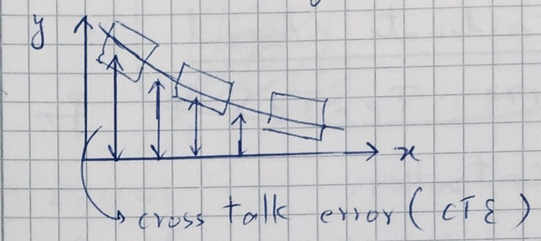
$$\boxed{\text{Error} = \frac{1}{N}\sum_{i=1}^{N}(y - (mx + b))^2}$$

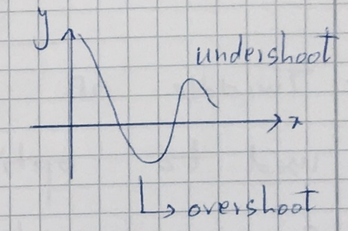
→ PID Controllers:

p → proportional

d → differential

i → integral





* In case of p controller:

$$\boxed{\text{steering}(\alpha) = -\tau_p(\text{cte})}$$

where:

τ_p → proportional coeff.

cte → y

* In case of pid controller:

$$\boxed{\alpha = -\tau_p(\text{cte}) - \tau_d \frac{d}{dt}(\text{cte})}$$

where $\frac{d}{dt} = \frac{\text{cte} - \text{old_cte}}{\Delta t} \longrightarrow 1$

* In case of pid controller:

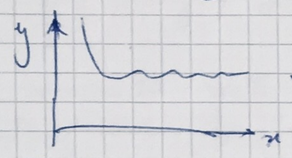
$$\boxed{\alpha = -\tau_p\,\text{cte} - \tau_d \frac{d}{dt}\text{cte} - \tau_i \int \text{cte}}$$

where $\int \text{cte} \rightarrow \Sigma \text{cte}$

* Systematic bias or drift:

for example wheels are unbalance & car drifts.

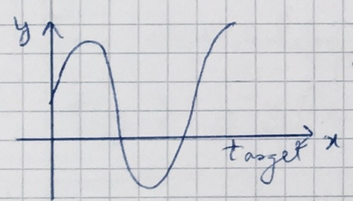
* In case of systematic bias:



→ p controller never reaches target b/c of drift.

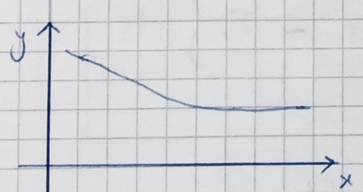
* Twiddle OR <u>Coordinate Ascent</u> is used to optimize τ_p, τ_d & τ_i for a given pid controller.

* p → helps move towards target.
* d → damps the oscillation.
* i → removes systematic bias.

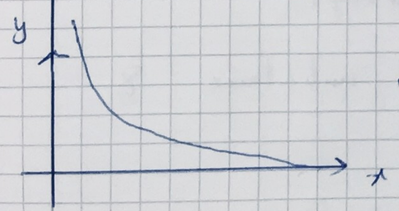


→ * w/o p controller
* starts by moving upward

* w/o d controller
* keeps crossing target w/ high oscillation

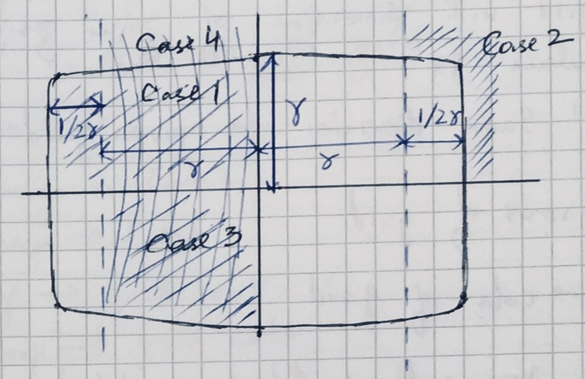


* w/o i controller
* systematic bias is not getting removed



* pid controller working correctly.

→ Cross Talk Error using radius



Case - 1

Condition : $x < r$

$$\boxed{cte = \sqrt{(x-r)^2 + (y-r)^2} - r}$$

case - 2:

condition : $x > 3r$

$$\boxed{cte = \sqrt{(x-3r)^2 + (y-r)^2} - r}$$

Case - 3:

$y < r$

$$\boxed{cte = 0 - y}$$

Case - 4:

$y > r$

$$\boxed{cte = y - 2r}$$

→ Algorithm for Dynamic programming

* value → grid with dimension of given grid values 99.
* policy → '' Same dimension as given grid
* Loop → rows of grid

  Loop → cols of grid

  if row, col matches goal(0), goal(1):

  value(row)(col) = 0

  policy(row)(col) = '*'

  change = True

  "Here we are doing recursion & starting from the goal."

  else grid(row, col) is free:

  Loop ~~in~~ → delta:

  "

  0 (x2, y2)

  1 (x2, y2) — (x, y) — (x2, y2) 3

  2 (x2, y2) "

  new_row = row + delta(index)(0)

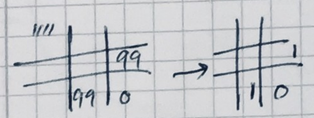
  new_col = col + delta(index)(1)

  check new_row & new_col are within boundary & grid is free:

  new_value = value(new_row, new_col) + cost

  if new_value < value(row)(col)

  "



  value(row)(col) = new_value

  policy(row)(col) = delta_name(index)

return value, policy

→ Algorithm — Stochastic Dynamic optimal Policy:

value → penalty, dimension as grid

policy → '', dimension as grid

change → True

while change is True:

change → False

Loop → grid rows:

Loop → grid cols:

if goal is row, col:

if value(row, col) > 0 → recursive → starting from goal

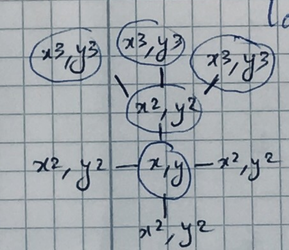
value(row, col) = 0

policy(row, col) = '*'

change = True

elif grid(row, col) is free.

loop → delta index



Loop → range(-1, 2) → Stochastic Loop

new_index = (index + stoch_index) % len(delta)

prob = success if stoch_index is 0 else failure

new_row = row + delta(new_index)(0)

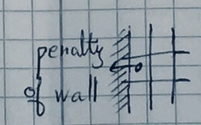
new_col = col + delta(new_index)(1)

if boundary & grid(new_row, new_col) is free:

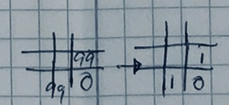
new_value = value(new_row, new_col) * prob + cost

else:

new_value = penalty * prob + cost



if new_value < value(row, col):



value(row, col) = new_value

policy(row, col) = delta_name(index)

change = True

return value, policy

→ Office Hours of PID Control:

* Twiddle:
  → Only good in 1D parameter optimization.
  → Susceptible to local minima b/c its a Hill climbing technique

* Smoothing & avoiding obstacle:
  → how we make sure a/f smoothing we dont hit obstacle which we avoid b/f smoothing

* When we design a controller, we guarantee that the controller works within Bayesian Attraction to find the boundary.

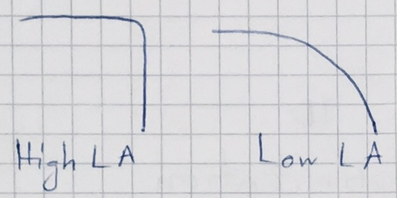
* olympiad
* Having focus on Learning something new.

* When a robot takes a turn on higher speed, then its Lateral acceleration will be high & it might be in d/f position compared to Low speed.

* Lateral acceleration (LA)
  how much turn on how much speed

so LA is small for large radius & high speed

but LA is high for small radius & normal speed



$LA \propto \frac{1}{\text{radius}}$

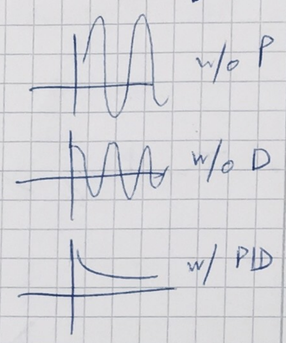
$LA \propto \text{speed}$

* Twiddle adjusts/optimizes 1 param at a time of τ_p, τ_d & τ_i.
* Twiddle will fail when we need to update > 1 param at a time
* Adaptive Sampler is also used in place of twiddle for parameter optimization of pid
* time lag b/w command & robot action, to handle it we use Predictive Controller & also the better modelling of the car like how much torque for how much steering movement.

Summary till here:

* Localization
  - histogram
  - markov model, InExact Movement
  - move, sense, Bayes Theorem
  - Total probability Theorem
* Tracking
  - Kalman filter
    - update & prediction Cycle
    - 1D (x)
    - 2D (x & $\dot{x}$)
    - 4D ($x, y, \dot{x}, \dot{y}$)
  - Particle filter
    - Resampling Wheel & Importance weight
* Search
  - Breadth first planning path finding, Left Turn policy
  - A* Heuristic path finding
  - Dynamic programming for optimal policy
  - Stochastic policy
  - path tracing
* pid Controller
  - Smoothing using Gradient Descent
  - Constrained Cyclic Gradient Descent
  - p, pd, pid Controllers
  - twiddle Algorithm for param optimization.
  - cross track error based on track radius.

	Continuous	Optimal	Universal	Local
Breadth first		✓		
A*		✓		
Dynamic Prog.		✓	✓	
Smoothing	✓			✓



P → Minimizes Errors

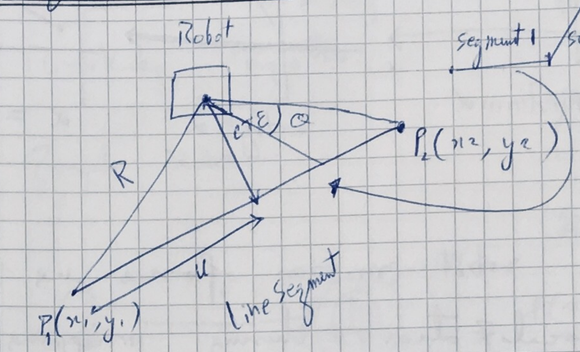
D → Avoids Overshoot

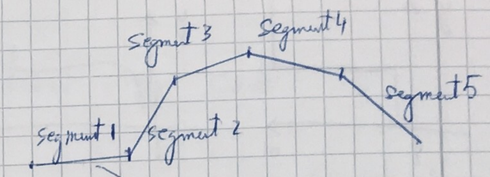
I → Minimizes Drift

damps overshoot

removes systematic bias

⇒ Segmented CTE





u → progress

u → how far robot has progressed

$$u = \frac{R_x \cdot \Delta x + R_y \cdot \Delta y}{\sqrt{\Delta x^2 + \Delta y^2}}$$

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

$R_x = x - x_1$

$R_y = y - y_1$

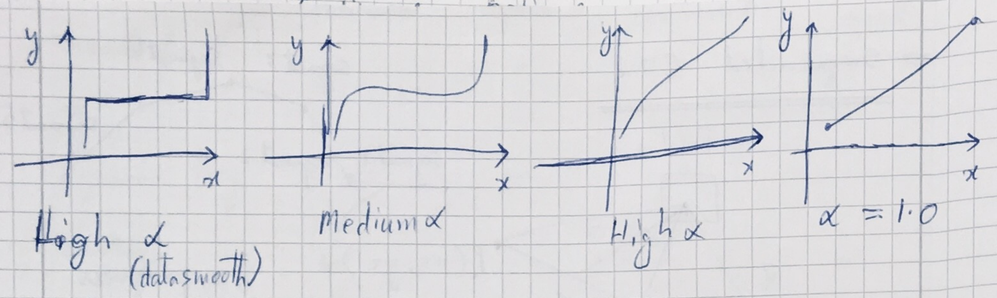
$$cte = \frac{R_y \cdot \Delta x + R_x \cdot \Delta y}{\sqrt{\Delta x^2 + \Delta y^2}}$$

where:

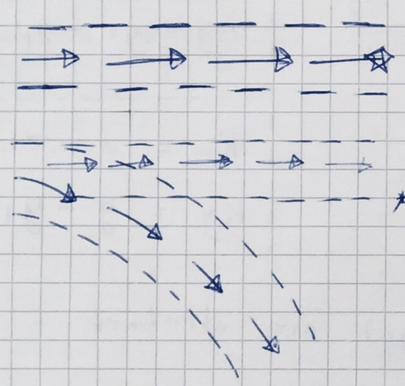
* (x_1, y_1) & (x_2, y_2) are attained using:

```
if(Len(path[index]):
    [x1, y1] = path(index)
    (x2, y2) = path(index+1)
```

* whenever u > 1 ; index += 1
* x, y are found using filter.getposition()



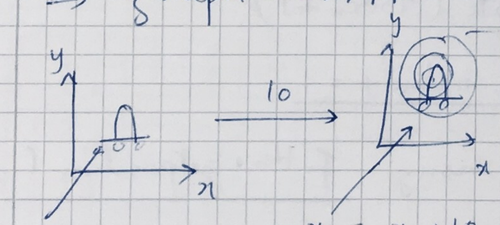
* Uncertainty in robot motion forces us to perform Localization during mapping.
  → considering SLAM



* no drift; so mapping will be correct

* robot has drift, so robot is not following correct path & map is not correct. Thats why we need Localization to reduce uncertainty.

→ Graph SLAM



→ We don't know exactly. instead we have Gaussian

$x_0 = 0$
$y_0 = 0$

$x_1 = x_0 + 10$
$y_1 = y_0$

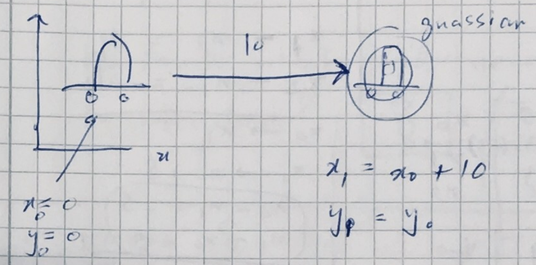
So: $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x_1 - x_0 - 10)^2/\sigma^2}$

$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(y_1 - y_0)^2/\sigma^2}$

Max Likelihood Criterion:
* We increase likelihood of x_1, y_1 given x_0, y_0 by reducing error using Gaussian Distribution.

→ robotics ai:
* ~~Slam~~ Understanding how the code is working together.

Graph SLAM:



$x_0 = 0$
$y_0 = 0$

$x_1 = x_0 + 10$
$y_1 = y_0$

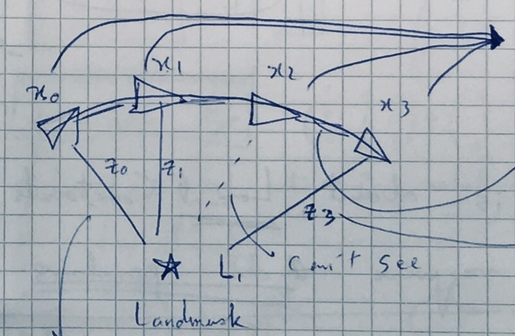
(Maximize likelihood of x_1, y_1 given x_0, y_0)

In Gaussian form

$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_1 - x_0 - 10)^2}{\sigma^2}}$

$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_1 - y_0)^2}{\sigma^2}}$

we increase likelihood by reducing uncertainty ← Max Likelihood Criterion
or error



* Initial Localization Constraints
* Relative Motion Constraint
* Relative measurement Constraints.

Landmark is seen by robot by relative measurement.

Quiz: 6 poses, 8 landmarks $6 \times 8 = 48$

So total constraints

Initial loc. = 1
Rel. motion = 5
Rel. measurement = 8

} 14 constraints

1
5
8

	x_0	x_1	x_2	L_0	L_1
x_0	1	-1			
x_1	-1	1	-1		
x_2		-1	1		
L_0					
L_1					

Matrix

-5	x_0
5	x_1
-4	x_2
	L_0
	L_1

Vector

diagonal → +ve

nondiagonal → -ve

	x_0	x_1	L_0	L_1
x_0	1	-1		
x_1	-1	$\boxed{2}$ -1	$\boxed{\cdot 1}$	
x_2		-1	1	
L_0		$\boxed{\cdot 1}$	$\boxed{\cdot 1}$	
L_1				

-5	x_0
5+4+(-9)	x_1 → 0
-4	x_2
9	L_0
	L_1

$x_0 \to x_1$

$x_1 \to x_0 + 5$

or

$x_0 - x_1 = -5$

$x_1 - x_0 = 5$

$x_1 \to x_2 - 4$

$x_2 = x_1 - 4$

$x_1 - x_2 = 4$

$x_2 - x_1 = -4$

$x_0 \to x_1$

$x_1 \to x_0 + 5$

$x_0 - x_1 = -5$ 1

$x_1 - x_0 = 5$

$x_1 \to x_2$

$x_2 \to x_1 - 4$

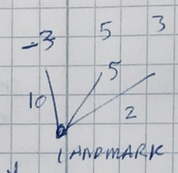
$x_2 - x_1 = 4$ 2

$x_1 - x_2 = 4$ 1

$x_1 : L_0$ distance 9

$x_1 - L_0 = -9$

$L_0 - x_1 = 9$



- Graph SLAM is about Local Constraints
- Graph SLAM requires add operations

- Every motion ties together with 2 locations
- Every measurement ties together 1 location w/ a landmark.

	x_0	x_1	x_2	L
x_0	1			
x_1				
x_2				
L				

	$x_0 = 0$

Initial Constraint
- Landmark

	x_0	x_1	x_2	L
x_0				
x_1				
x_2				
L				

$x_0 + 0 - x_1$

$x_0 : L$

$x_2 : L$

no direct relation b/w x_0 & x_2

	x_0	x_1	x_2	L
x_0				
x_1				
x_2				
L				

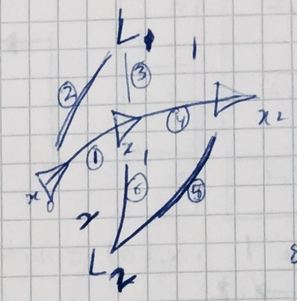
$x_0 - x_1$

$x_1 - x_2$

$x_0 - L$

$x_2 - L$

	x_0	x_1	x_2	L_1	L_2
x_0	/	/	O	/	O
x_1	/	/	/	/	/
x_2	O	/	/	O	/
L_1	/	/	O	/	O
L_2	O	/	/	O	/



No relation

1 $x_0 : x_2$

2 $L_1 : x_2$

3 $x_0 : L_1$

4 $L_1 : L_2$

Relation

$x_0 : L_1$

$x_1 : L_1$

$x_0 : x_1$

$x_1 : x_2$

$x_1 : L_2$

$x_2 : L_2$

Mark which will never be touched? $\boxed{8}$

$\begin{pmatrix} -3 \\ 5 \\ 3 \end{pmatrix} \to \begin{pmatrix} -8 \\ -2 \\ 3 \end{pmatrix}$

$\begin{pmatrix} -8 \\ -2 \end{pmatrix}$

⇒ omega ξ, x_i

$\boxed{\mu} = \boxed{\Omega} \boxed{\xi}$

$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$\begin{pmatrix} -3 \\ 5 \\ 3 \end{pmatrix}$

move 1

move 2

$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$

$\begin{pmatrix} 1 & -1 & \\ -1 & 1 & \\ 0 & 0 & 0 \\ & -1 & \end{pmatrix}$

$\begin{pmatrix} -3 \\ 5 \\ 3 \\ -8 \\ -2 \\ 3 \end{pmatrix}$

$\boxed{\mu = \Omega^{-1} \xi}$

$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$

↳ best estimate for robot locations & landmark locations.

	x_0	x_1	x_2
x_0	+1 -1		
x_1		3x3	
x_2			

-3	x_0
3x1	x_1
	x_2

initial position

$x_0 = -3$

$x_1 = x_0 + 5$ — move 1

$x_2 = x_1 + 3$ — move 2

$\mu = (\Omega^{-1})\xi$

$x_1 - x_0 = 5$

$x_0 - x_1 = -5$

$x_1 - x_2 = -3$

$x_2 - x_1 = 3$

$\begin{pmatrix} -3 \\ 5 \\ 3 \end{pmatrix}$

	x_0	x_1	x_2
x_0	1+1	-1	
x_1	-1	1+1	-1
x_2	0	0-1	0+1

	2	-1	0
	-1	2	-1
	0	-1	1

-3-5+0	x_0
0+5-3	x_1
0+0+3	x_2

-8
-2
3

$10 - 3 = 7$

-7

$-13 + 10 = -3$

$\Omega^{-1}\xi$

-13×5

	x_0	x_1	x_2
x_0	0,0 / 1	0,1	0,2
x_1	1,0	1,1	1,2
x_2	2,0	2,1	2,2

0
−3
0
0

$\Omega(0)(0) = 1$; $x_i(0) =$ initial

loop for $0 \to 2$

Ω (outer)(inner) += 1

Ω [outer][inner + 1] += 1

x_i[outer] += − move(outer index)

x_i(inner + 1) += move(outer-index)

$x_0 = -3$

$x_1 = x_0 + 5$

$x_2 = x_1 + 3$

$$\boxed{\begin{aligned} x_0 - x_1 &= -5 \\ x_1 - x_0 &= 5 \end{aligned}}$$

$$\boxed{\begin{aligned} x_1 - x_2 &= -3 \\ x_2 - x_1 &= 3 \end{aligned}}$$

initial

$x_0 = -3$

move 1 → 5

x_0 , x_1 , 5

0,1

x_1 → 0,1 / 1,0

(x)(y) =

(outer)(inner)

equation 1:

$$\begin{bmatrix} -3 \\ 5 \\ 3 \end{bmatrix} \begin{matrix} 0 \\ 1 \\ 2 \end{matrix}$$

~~for exp~~

for 0 , 0 0,0 (0, 1)

1+1	−1		

−3 − 5

0,0

1+1 −	−1 +1	
1	−1	+1
0	−1	

−3 − 5
5

outer = 0

Ω (outer)(0) += +1

Ω (outer (1)+= −1

~~Ω (outer)(2)~~

(x_i)(outer+1) += − move(outer)

(x_i)(outer+1) += + move(outer)

outer = 1

Ω (1)(0) += −1

Ω (1)(1) += +1

Ω (1)(2) += −1

x_i (2) = −3

x_i (2) = +3

outer = 2

Ω (2)(0)+= −1 ×

Ω (2)(1) += −1

Ω (2)(2) += 1

x_i(2+1) doesn't exist

(start)(Len −1) ≠ free

(Len −1)(start) ≠ free

when Loop ↑

sign * = −1

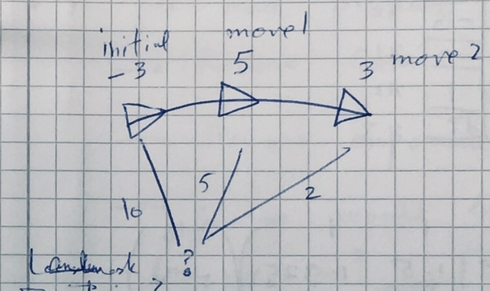
omega(0)(0) = 1 ;

x_i (0) = initial

$$\Omega = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix} \longrightarrow \Omega$$

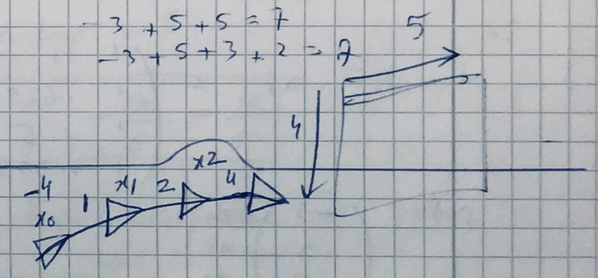
$$\begin{pmatrix} -3 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -5 \\ 5 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -3 \\ 3 \end{pmatrix} \longrightarrow \xi$$



~~Sum~~: 10 − 3 = 7

−3 + 5 + 5 = 7

−3 + 5 + 3 + 2 = 7



Example:

$x_0 \longrightarrow -4$

$x_1 \longrightarrow x_0 + 1$

$x_2 \longrightarrow x_1 + 2$

$x_3 \longrightarrow x_2 + 4$

find M :

$$\Omega = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

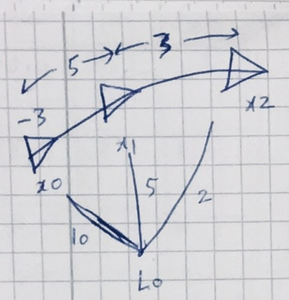
$$x_i = \begin{pmatrix} -4 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -2 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -4 \\ 4 \end{pmatrix} = \begin{pmatrix} -5 \\ -1 \\ -2 \\ 4 \end{pmatrix}$$

And:

$$\Omega^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}$$

So:

$$M = \Omega^{-1} \cdot x_i = \begin{pmatrix} -4 \\ -3 \\ -1 \\ 3 \end{pmatrix} \begin{matrix} - x_0 \\ - x_1 \\ - x_2 \\ - x_3 \end{matrix}$$



$x_0 = -3$

$x_1 \rightarrow x_0 + 5 \rightarrow$ move 1

$x_2 \rightarrow x_1 + 3 \rightarrow$ move 2

$L_0 \rightarrow x_0 + 10 \rightarrow z_0$

$L_0 \rightarrow x_1 + 5 \rightarrow z_1$

$L_0 \rightarrow x_2 + 2 \rightarrow z_2$

$\Omega =$

	x_0	x_1	x_2	L_0
x_0	1+1+1	-1		-1
x_1	-1	1+1+1	-1	-1
x_2		-1	1+1	-1
L_0	-1	-1	-1	1+1+1

-3-5-10	x_0
5-3-5	x_1
3-2	x_2
10+5+2	L_0

$\leftarrow x_i$

$\Omega =$

	x_0	x_1	x_2	L_0
x_0	3	-1	0	-1
x_1	-1	3	-1	-1
x_2	0	-1	2	-1
L_0	-1	-1	-1	3

; $x_i =$

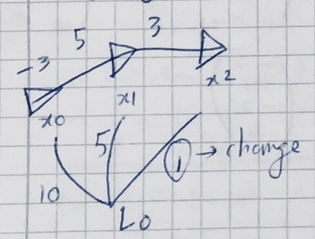
-18	x_0
-3	x_1
1	x_2
17	x_3

So:

$$M = \Omega^{-1} \cdot x_i = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1.625 & 1.5 & 1.375 \\ 1 & 1.5 & 2 & 1.5 \\ 1 & 1.375 & 1.5 & 1.625 \end{pmatrix} \begin{pmatrix} -18 \\ -3 \\ 1 \\ 17 \end{pmatrix}$$

$$M = \begin{pmatrix} -3 \\ 2 \\ 5 \\ 7 \end{pmatrix} \begin{matrix} \rightarrow x_0 \\ \rightarrow x_1 \\ \rightarrow x_2 \\ \rightarrow L_0 \end{matrix}$$

$\Rightarrow$ Changing path to:



no change, Absolute

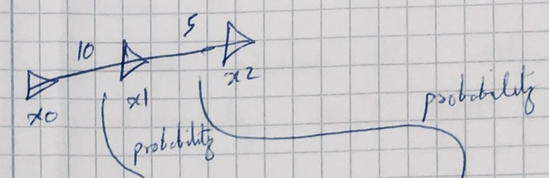
Then $M = \begin{pmatrix} -3 \\ 2.125 \\ 5.5 \\ 6.875 \end{pmatrix}$ } Relative $x_i = \begin{pmatrix} -18 \\ -3 \\ 2 \\ 16 \end{pmatrix}$

* So, initial position is the only <u>absolute constraint</u>, Motion & Measurement Constraints are relative (Landmark)

* Using $\Omega^{-1}\xi$, we find optimal result of rubberband

relaxation.

$\Rightarrow$ Confidence Measurement:



$$f_1(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_1 - x_0 - 10)^2}{\sigma^2}} \qquad f_2(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x_2 - x_1 - 5)^2}{5^2}}$$

And:

$$f(x) = f_1(x) \cdot f_2(x)$$

* To get max. product:
  - Remove Const. $\frac{1}{\sqrt{2\pi\sigma^2}}$.
  - Remove exponential i.e $\exp \frac{-1}{2}$
  - So we would end up with:

$$\frac{(x_1 - x_0 - 10)^2}{\sigma^2} \longrightarrow \boxed{\frac{x_1}{\sigma} - \frac{x_0}{\sigma} = \frac{10}{\sigma}}$$

→ Here $\frac{1}{\sigma}$ tells how confident we are with the measurement.

→ $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -\Delta \\ \Delta \end{pmatrix} \xrightarrow{\text{should be}} \begin{pmatrix} x & -x \\ -x & x \end{pmatrix} = \begin{pmatrix} -\Delta, x \\ \Delta, x \end{pmatrix}$

where x is a Large number

⇒ Exam Questions:

Question # 01:

1 dice shows 6, other is hidden
probability that both show 6?

Sol.

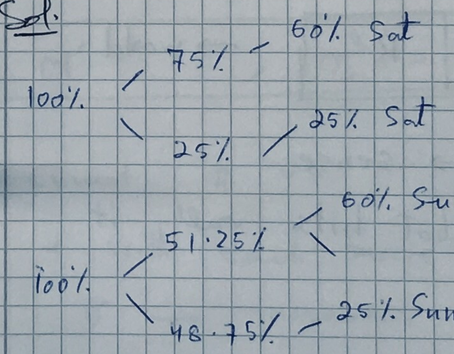
$P = P(\text{dice } 1) \cdot P(\text{dice } 2)$

$P = (1)(1/6) = 1/6$

Question # 02:

- 60% chance it'll rain next day too.
- 25% chance it'll rain next day too if its not raining today.
- today (Fri) rains 75% chance.
- Probability that atleast one day of weekend it won't rain?

Sol.



$\Rightarrow P(\text{Sat}) = (75\%)(60\%) + (25\%)(25\%)$

$\Rightarrow P(\text{Sat}) = 51.25\%$

$\Rightarrow P(\text{Sunday}) = (51.25\%)(60\%) + (25\%)(48.75\%)$

$\Rightarrow P(\text{Sunday}) = 42.9375$

No.	Fri	Sat	Sun
	0	0	0
6	0	0	1
5	0	1	0
4	0	1	1
	1	0	0
3	1	0	1
2	1	1	0
1	1	1	1

$P(6) = P(3) = 0.1218$

$P(5) = P(2) = 0.5125$

$P(4) = P(1) = 0.3075$

So:

$P = \dfrac{2(0.1218) + 2(0.5125) + 2(0.3075)}{6}$

$P = 0.3139$ of rain

$\boxed{P = 0.6861 \rightarrow \text{no rain atleast 1 day}}$

Que #03:

G	G	R	G	R

→ world

$p = \{$ Uniform distribution $\}$

measurement → red.

Error $\{ \rightarrow 0.1$

find:

Sense function = ?

Sol.

pMiss = 0.1 ; pHit = 0.9

$p = \{0.2, 0.2, 0.2, 0.2, 0.2\}$

Then 2nd sense fn. with normalization

Result = $\{0.0476, 0.0476, 0.4285, 0.0476, 0.4285\}$

Que # 04:

G	G	R	G	R

→ world

* Sense → move right → sense
* not cyclic, so if robot hits wall, it stays there.

$\boxed{R}$ → measurement

Sol.

→ Sense

0.0476	0.0476	0.4285	0.0476	0.4285

Now

x 0.1 if right is green

x 0.9 if right is red

x 0.1 or 0.9 if hits wall

green → red

Here we don't see current position but the position on the right as robot move 1 right.

So

0.0476	0.0476	0.4285	0.0476	0.4285
x 0.1	x 0.9	x 0.1	x 0.9	x 0.9

→ red & boundary

right is green | right is red | right is green | right is red

Quest. # 05:

$\mu_1 = 1$; $\sigma_1^2 = 1$; $\mu_2 = 3$; $\sigma_2^2 = 1$

Sol.

Since measurement is done:

$$\mu' = \frac{\mu_1 \sigma_2^2 + \mu_2 \sigma_1^2}{\sigma_1^2 + \sigma_2^2} = \frac{(1 \times 1) + 3(1)}{1 + 1}$$

$$\sigma_1^2 = \frac{1}{1/\sigma_1^2 + 1/\sigma_2^2} = 1 \Big/ \left(\frac{1}{1} + \frac{1}{1}\right)$$

$$\boxed{\mu' = 2 \quad ; \quad \sigma_1^2 = 1/2}$$

Quest. # 06:

* With each move:

σ → increases

μ → adjust (changes)

Quest. # 07:

* Particle filters are Multimodal, Continuous.
* They scale exponentially

Question # 09:

For admissible heuristic:

$h(x,y) \leq cost$

For example:

cost = 2 & $h(x,y) = 2$ So: $h(x,y) \leq cost$

0	1	
3	2	

→ admissible

cost = 2; $h(x,y) = 10$ so $h(x,y) \not\leq cost$

0	1	10
	10	
10		

→ not admissible

given: cost = 2; $h(x,y) = 5$

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

* So its not admissible heuristic if cost = 1
* Can only be admissible heuristic if cost is here.
* Note that's why we start from goal to start point & goal is 0.

Question # 10:

	C		A	
	B		D	
		Car		

forward, right → 1 cost

left → 14 cost

Sol.

Cost A = 6 ; Cost B = 14; Cost C = 19

Cost D = 4 .