Sebastian Thrun

- planing, perception, control

- Team: Anthony Navarro
  Andy Brown
  Cezanne Camacho
  Elica White
  Tarian Ziyaee
  Jessica Lulovics
  Erika Alonzo

David Silver
↳

- map
- Localization
- plan ⟶ choose a trajectory amongst all possibilities.

- shortest path
(split) Decision
- search problem

- Darpa Grand Challenge

- Red Whittekers
  Sebastian Thrun
  Mike Montemerlo
  Anthony Levandowski

$$a_{lat} = \frac{V^2}{R}$$

lateral acceleration:

In the Junkyard conditions the maximum lateral acceleration that Carla's tires support is 12 m/s^2. If she travels at a velocity of 30 m/s what is the sharpest turn (minimum turning radius) she can make?

75 m

129, 30 → stop

125, 40 → full gas back, full steer right

till 126,5 , 34 full gas back, full steer left

- noise (not deterministic, every run results in a slightly different behavior)
- no accuracy down to desired level. (you say stop at position 40, it actually stops at 38)
- deficult to test
- the less parameters, the better (e.g. considering x and y at the same time makes it more deficult
- update rate of information is very important
- windows of action ranges of values with probability.

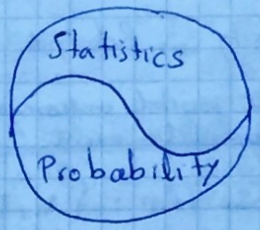
Bayes Rule:

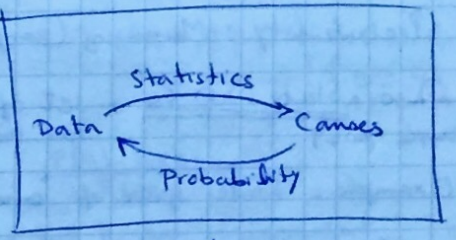
- Normal robots needs to know everything about the world. When the world is wrong, then it would do fatal mistakes.
- Bayes Rule permits for uncertainty; that the robot might not know and might have to explore the world in order to get knowledge about it.

- conditional probability
- Bayes Rule
- Probability distribution (+ Bayes rule) ⇒ is crucial for robot localization
- robot localizer
  - Proj 1: 2D Histogram filter.

Uncertainty

- Uncertainty / control noise
- "knowable" in probabilistic sense vs. # certain
- uncertain → Nothing is certain! in a SDC
  - what traffic around the car does in near future
  - location of the car
  - speed of the car
  - Steering
  - ...
  - ...
- complete certainty vs. sufficient certainty
- A car is an imperfect mechanical system
- Our brain is a high performance adaptive controller





Probability: Gives us the language for describe/discuss the relationship between Data ↔ Causes data and the underlying causes.

Let's "make data" by flipping the coin:

Data = {H, H, H} ⇝ Data from three flipps (measurement)

Probability is a method of describing the anticipated outcome of these coin flipps.

- Fair Coin vs. Loaded Coin

$$P(A) = 1 - P(\neg A)$$

- A coin flip is an example of a "Probabilistic Event"
- Probabilistic Event: a set of outcomes to some experiment where each outcome has a probability.
- Example: Coin Flipping
  - outcomes: H, T
  - P(H) = 0.5, P(T) = 0.5

- Truth Table: listing e.g. all possible outcomes of a 2 coin flip

Flip 1	Flip 2
H	H
H	T
T	H
T	T

Probability: Managing Complexity in Robotics

Ω • **State Space** (Sample space): set of all possible outcomes for a probabilistic event.

Example: State space of "one" coin flip: $\{H, T\}$

- Calculation needed for N events each having a space state of size n is n^N
  → Exponential complexity growth

- Probability $\approx$ $\frac{\text{Number of ways an event can occur}}{\text{Total number of events that could occur}}$

Conditional Probability:

the basic mechanism that a SDC can take some sensor data an infere something else. For example it could see a picture of a tree and infere "I am about to crash". The language that is needed in mathematics to infere from sensor measurements to some internal belief is called Conditional probability.

- Frequent sensor measurement ⇝ more "Fresh" data
  ⇝ better estimate of conditional probability
  $P(\text{Event} \mid \text{measurement})$
- "Cloudiness now" and "cloudiness one minute from now" are not independent events.
- We take sensor measurements to use that data to inform our probabilistic estimate of an event.
- (Intuition) Conditional Probability:

> Taking advantage of what we know, to make better estimates about what we don't.

$P(A \cap B)$:

$P(A,B)$ = Probability of → [A and B] $A \cap B$

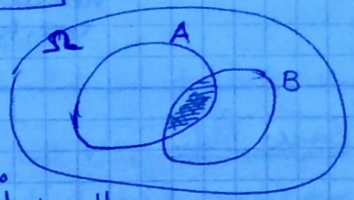
$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

If A and B independent:

$$P(A \mid B) = \frac{P(A,B)}{P(B)} = \frac{P(A) \cdot P(B)}{P(B)} \Rightarrow$$

$P(A \mid B) = P(A)$ if A and B are independent

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$ **Intuition:**



Intuition: Restricting the sample space to situations in which B occurs; Because if possible outcomes for A and B are restricted to those in which B occurs, this set serves as the new sample space

$$P(A \mid B) = \frac{P(A,B)}{P(B)} \qquad P(B \mid A) = \frac{P(A,B)}{P(A)}$$

$$\Rightarrow P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$$

$$\Rightarrow P(B \mid A) = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

Cancer and Test example from lecture: C: Cancer; Test: P or N

$P(C) = 0.9$
$P(\neg C) = 0.1$

$P(C) \rightsquigarrow P(\neg C)$

$P(P \mid C) \rightsquigarrow P(N \mid C)$ — $P(P \mid C) = 0.9$, $P(N \mid C) = 0.1$

$P(P \mid \neg C) \rightsquigarrow P(N \mid \neg C)$ — $P(P \mid \neg C) = 0.2$, $P(N \mid \neg C) = 0.8$

Total Probability: ← Total Probability

$$P(P) = P(P \mid C) \cdot P(C) + P(P \mid \neg C) \cdot P(\neg C)$$

Truth Table

Cancer	Test	
C	P	$P(P,C) = 0.09$
C	N	$P(N,C) = 0.01$
¬C	P	$P(P,\neg C) = 0.18$
¬C	N	$P(N,\neg C) = 0.72$

$\Sigma = 1$

Law of Total Probability:

Example: Coin w T, H; P(T) = P(H) = 1/2
1. flip → T? do nothing
2. flip → H? → flip again.
what is the prob. of H at second flip?
$P(H^2) = P(H^2 \mid T^1)P(T^1) + P(H^2 \mid H^1)P(H^1)$
$= 0 \cdot 1/2 + 1/2 \cdot 1/2 = 1/4$

If $B_1, B_2, B_3, \ldots$ is a partition of a sample space S, then for any event A:

$$P(A) = \sum_i P(A \cap B_i) = \sum_i P(A \mid B_i) * P(B_i)$$

(think of it as: calculating the sum of all probabilities necessary to ensure all scenarios for a specific event are included.)

Bayes Rule

Reducing Uncertainty

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

$$P(C_k \mid x) = \frac{P(C_k) \cdot P(x \mid C_k)}{P(x)}$$

$$\text{posterior} \propto \frac{\text{Prior} \cdot \text{likelihood}}{\text{evidence}}$$

- SLDs make a lot of measurements for example about speed, location of the car that are not certain. These measurements are dependent on one another
- Bayes rule gives us a tool to combine the (on each other depending information) from different sensors and combine them into a more reliable representation of the car's position, movement, and environment.

↓ using conditional probabilities

Bayes Rule
Reducing Uncertainty

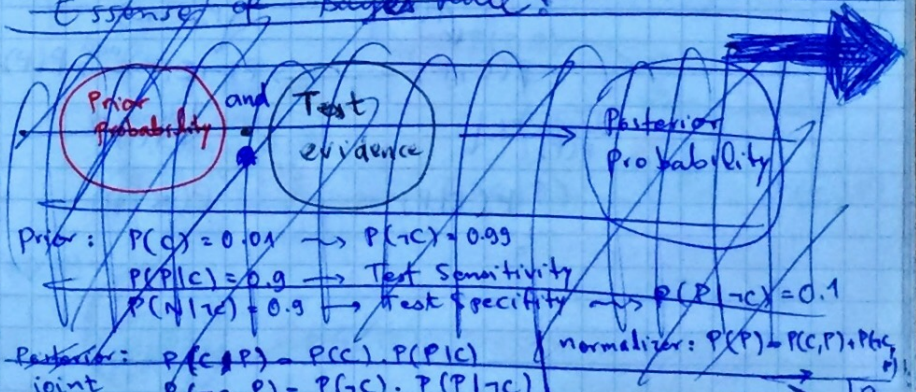
- Bayes rule is very important in robotics.
- Bayes rule summarized in one sentence:

> "Given an initial prediction, if we gather additional related data, data that our initial prediction depends on, we can improve that prediction"

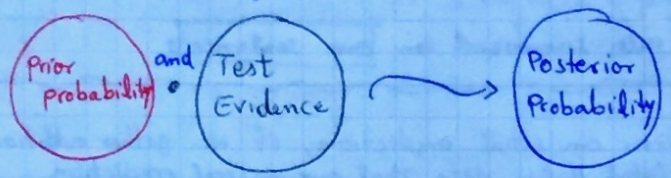
- initial prediction ≈ prior belief
- The sensor data combined with what we already know about the road and the car gives us more information about where are location is most likely to be
- Bayes rule gives us a mathematical way to correct our measurements and lets us move from an uncertain prior belief to something more and more probable.

Cancer Test: Sensitivity ≈ $P(P \mid C)$
Specificity ≈ $P(N \mid \neg C)$

Essence of Bayes Rule:



Essense of Bayes rule:



Prior: $P(C) = 0.01 \rightsquigarrow P(\neg C) = 0.99$
$P(P|C) = 0.9$
$P(N|\neg C) = 0.9 \rightsquigarrow P(P|\neg C) = 0.1$

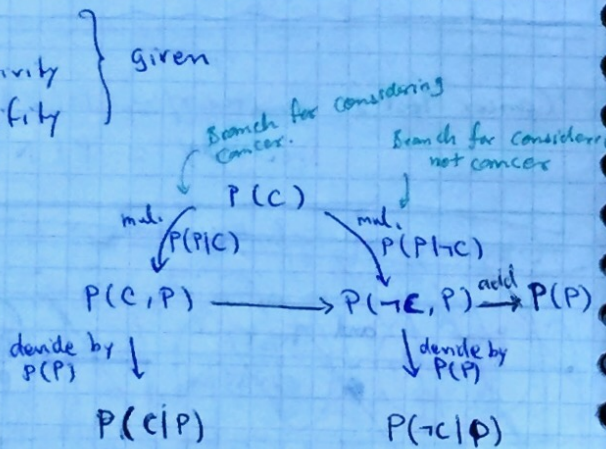
joint: $P(C,P) = P(C) \cdot P(P|C) = 0.009$
$P(\neg C,P) = P(\neg C) \cdot P(P|\neg C) = 0.099$

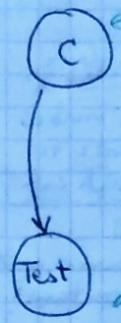
normalizer: $P(P) = P(C,P) + P(\neg C,P) = 0.108$

posterior: $P(C|P) = \frac{P(C,P)}{P(P)} = 0.0833$ $P(\neg C|P) = \frac{P(\neg C,P)}{P(P)} = 0.916$ ($\Sigma = 1$)

$P(C)$ prior
$P(P|C)$ test sensitivity
$P(N|\neg C)$ test specifity
} given

recieved: P





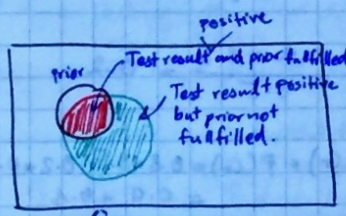
Hidden variable
we care about, but we can't measure

Prior: tells us how frequent this variable is

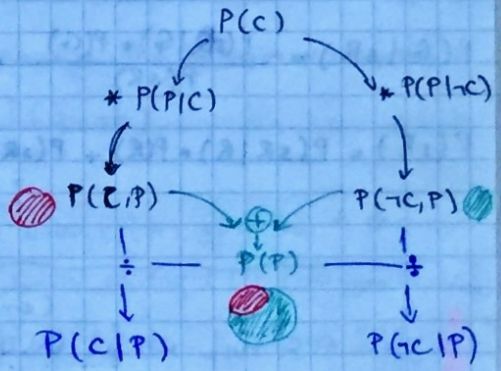
A Test characterized by
how often it is positive when the variable is true
and how often it is negative when the variable is false

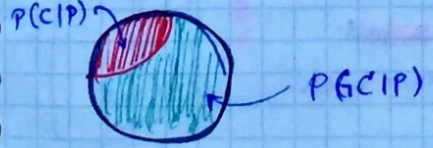
likelihood
sensitivity $P(Pos|C)$
specifity $P(Neg|\neg C)$

Bayes rule now takes the prior, multiplies in the measurement frequencies to give us a new variable $P(C,P)$ and the same for measurement given the opposite assumption about the hidden variable, giving us $P(\neg C,P)$, we add those to come to $P(P)$ normalizer and we devide $P(C,P)$ and $P(\neg C,P)$ by normalizer to arrive at our best estimate of the hidden variable C given our test result.



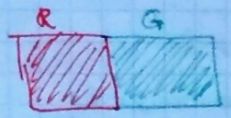
But the red and green areas don't add up to 1! therefore divide by total area





Example:

A robot living in a world with two cells: Red and Green



The robot makes measurements to see in which cell he is.

But measure ments are not certain!

$P(R) = P(G) = 0.5$

$P(\text{see Red} \mid \text{at Red}) = 0.8 \rightsquigarrow P(sG|R) = 0.2$

$P(\text{see Green} \mid \text{at Green}) = 0.8 \rightsquigarrow P(sR|G) = 0.2$

sR: see red

Assume Robot sees red.

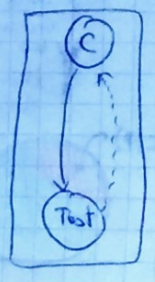
- What is now the posterior porobability that the robot is at red cell? = 0.8 $P(R|sR)$
- What is the probability that it is in green cell even though it sees red? 0.2 $P(G|sR)$

$$P(R|sR) = \frac{P(sR|R) * P(R)}{P(sR)} = \frac{0.8 * 0.5}{0.5} = 0.8$$

$$P(G|sR) = \frac{P(sR|G) * P(G)}{P(sR)} = \frac{0.2 * 0.5}{0.5} = 0.2$$

$$P(sR) = P(sR|R) * P(R) + P(sR|G) * P(G) = 0.8 * 0.5 + 0.2 * 0.5$$
$$= 0.4 + 0.1$$
$$= 0.5$$

We are now able to consider a hidden variable and understand how a test can give us information back about it



Steps to Bayes Rule:

H = Hypothesis
T = Test

1. **Prior Probabilities**: Based on previous data, how likely is that H happens?

$$P(H) \rightsquigarrow P(\neg H)$$

2. **Conditional / Test Probabilities**:

Based on sensor or test data collection, how likely is that a certain test or sensor reading is to occur, given that the hypothesis H does or has not occured?

$$P(T|H) \rightsquigarrow P(\neg T|H)$$
$$P(T|\neg H) \rightsquigarrow P(\neg T|\neg H)$$

3. **Joint Probabilities**:

of the prior and the test:

$$P(T, H) = P(T|H) \cdot P(H)$$

likewise: $P(T, \neg H) = P(T|\neg H) \cdot P(\neg H)$

4. **Total Probabilities**:

- Total probability of a test result or sensor reading.
- This is the sum of joint probabilities in which that test result occurs
- We need this to normalize the posterior probability

$$P(T) = P(T, H) + P(T, \neg H)$$

5. **Posterior Probability**:

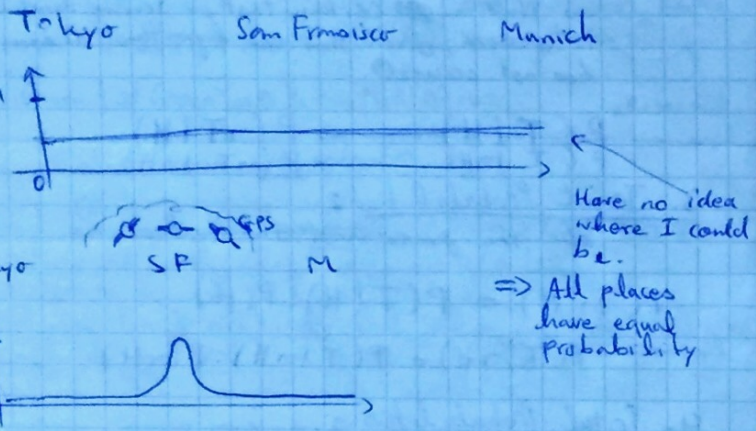
Probability of an event (H) given the a certain test or a certain sensor reading (T) [Bayes Rule]

$$P(H|T) = \frac{P(T|H) \cdot P(H)}{P(T)} \qquad P(H|T) = \frac{P(T, H)}{P(T)}$$

$$\text{posterior} \propto \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

Probability Distributions:

- A way in which a SDC represents its internal Beliefs about the world.
- A mathematical way to represent uncertainty across all possible outcomes



Have no idea where I could be.

⇒ All places have equal probability

- Very useful in visualizing probabilities.
- Can be worke with using algebra, Calculcus, linear algebra
- Probability distribution allows us to show the probability of an event using a mathematical formula.

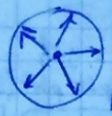
Two Types:

- Discrete probability distributions — For discrete variables like coin flips, dice rolls, locations on a grid, ...
- Continous probability distributions — for continous variable like height, time, ...

- I Discrete prob. dist.:
  - the variable can take discrete values
  - all values of y are greater or equal to 0
  - for each x, P(x) = y
  - Sum of all y values is 1

Continous Prob. dist:

! Every outcome (exact outcome) has prob. 0 !!

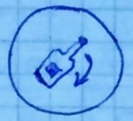


P(angel = 180) = 0

P(angle = 179.999) = 0

Every outcome has P = 0 !

Because it is impossible to have the outcome be exactly equal the given value!



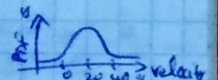
X: outcome

P(0 < x < 180) = 0.5

P(260 < x < 290) = 1/12 = 0.08333

P(179 < x ≤ 180) = 1/360 = 0.00277

- Probability Density Function PDF



- Characteristics of continuous prob. dist. and Prob. density func
  - y values are greater or equal zero
  - prob. of specific x occuring is equal zero
  - prob. of an event occuring between two values of x, is equal the area under the curve between x and x
  - The total area under the prob. dens. func. is 1
- For a continuous dist. we can only calculate the probs.
- between a range of values. !

Uniform Continuous Dist.:

- PDF ⇝ an equation that mathematically represents a continous distribution.

$P = \int PDF$

Densities can be larger than 1

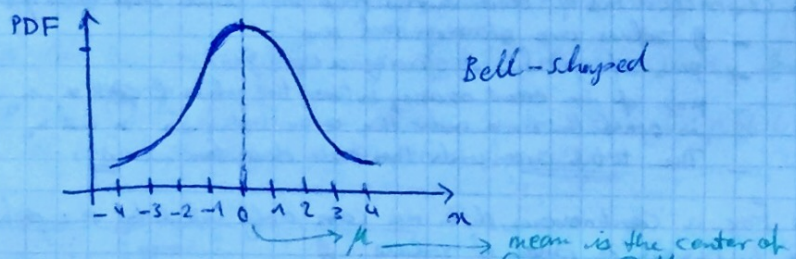
But the sum under the continous curve must add up to 1

```
import matplotlib.pyplt as plt

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

plt.scatter(x,y) // plt.bar(x,y) // plt.plot(x,y)//
plt.xlabel("x values")
plt.ylabel("y values")
plt.title("X values versus Y values")
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11])
plt.show()
```

```
plt.plot([0,360],
[y,y],
"--",
color=-)

plt.imshow(grid,
cmap=gray
clim=both)
```

Gaussian Distributions

- There are many different continuous prob. dists.
- For each application, we use a suitable dist
- We use Gaussian dist. to model uncertainty in sensor measurments.



Bell-shaped

→ mean is the center of Gaussian Bell.

PDF for Gaussian dist:

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

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

→

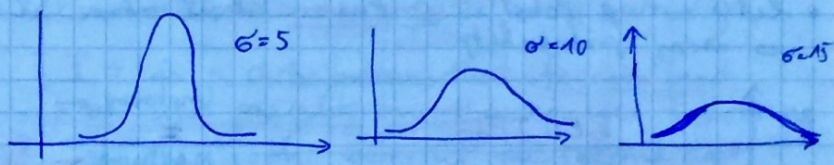
μ: mean of population

σ: standard dev. of population

- μ: shifts the curve along the x axis.
- σ: makes the curve taller or wider.

  ⇒ As the standard deviation increases, the curve becomes wider and flatter,

  ⇒ the uncertainty increases



Central limit Theorem:

- Says: → for independant, random variables

  If you get large enough samples from a population and then calculate the sample's mean, these means will be normally distributed

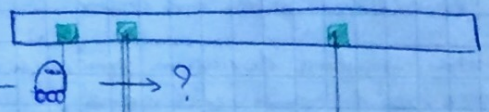
Robot Localization:

- GPS → 2-10 m accuracy
- For a SDC → we need 2-10 cm
- The techniques here require a good map. If we don't have a map, we have to do SLAM.

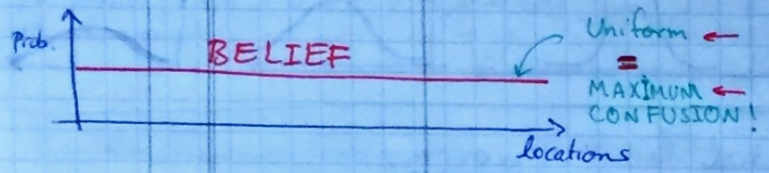
→

Imagine a 1D world

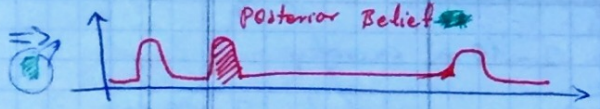
= Door



- let's map robot's confusion about where it is using probability



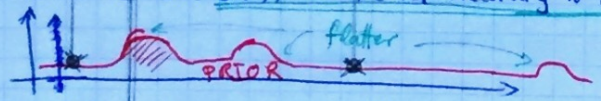
- to be able to localize we have to use world's distinctive features.
- let's assume our world has 3 distinctive land marks → 3 doors, all similar and non-distinguishable
- let's assume robot senses a door ⇒ new Belief function



Our current best Belief of where we are

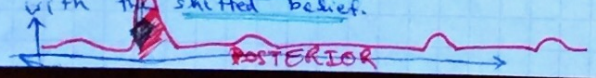
Posterior Belief: the belief after a measurement has been taken!

- now let's assume robot moves forward to the right. Then we can shift the Belief according to motion.



CONVOLUTION

- Now Robot measures again a door. Now we multiply our belief, that is prior to the second measurement with the shifted belief.

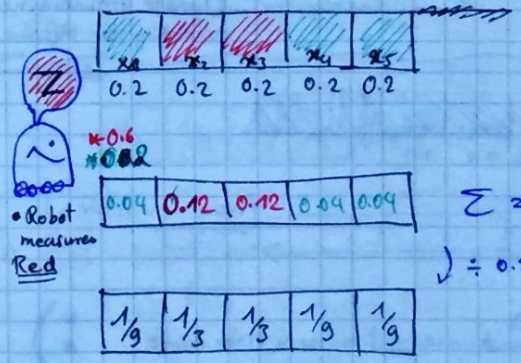


- The peak in the resulting function is the only place in the prior that corresponds to the measurement of a door, whereas all the other places of door have a lower prior belief.
- As a result, this function is really interesting.

  It is a distribution that focuses most of its weight on the correct hypothesis of the robot being at the second door.

  And it provides very little belief to places far away from doors and other doors.

Robot localized itself!



$P(X_i|Z)$ posterior distribution

- The measurement update function is the absolute key function of localization

↳ Takes in a measurement and a probability array, and calculates a new probability array based on the measurement.

posterior ←

```
sense(p, Z):
    q = []
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
```

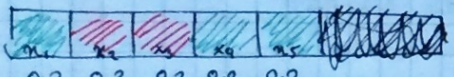
p = [0.2, 0.2, 0.2, 0.2, 0.2]
world = [G, R, R, G, G]
pHit = 0.6
pMiss = 0.2

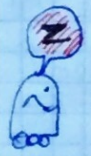
another way:

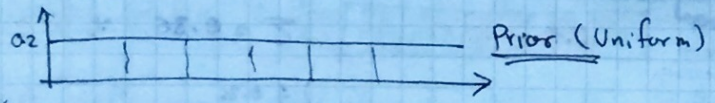
```
q = []
for i in range(len(p)):
    if Z == world[i]:
        q.append(p[i] * pHit)
    else:
        q.append(p[i] * pMiss)
```

Measurement Update Function

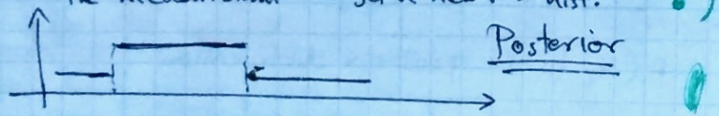
Update probabilities according to measurement







(by incorporating the measurement we get a new prob. dist:



:)

!

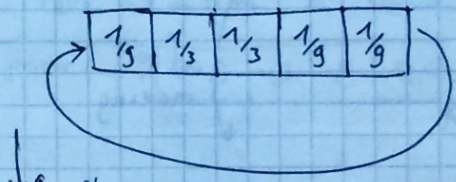
→

- Now for multiple measurements:
  we just recursively update our probability vector using new measurements and its prior value:

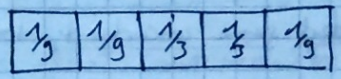
```
p = [0.2, 0.2, 0.2, 0.2, 0.2]
for m in measurements:
    p = sense(p, m)
```

Robot Motion:

- As robot moves, the prior probabilities(?) shift
  (if the robot movement is perfect:
  i.e. it moves exactly as far as it thinks it does)



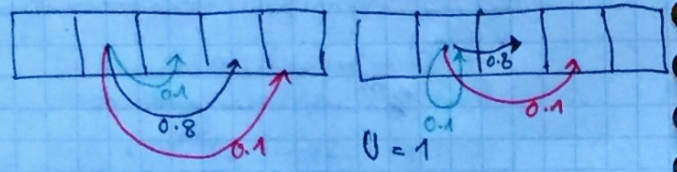
One step to the right



- But, robot movement is inaccurate (not perfect)
  - we have to model this
  - this is the primary reason, why localization is hard.

→

Inaccurate Robot Motion:



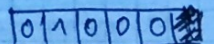
- The robot should move $U=2$ cells. It does correct with $p=0.8$. But with $p=0.1$, it just move too few or with $p=0.1$ it moves too far, overshooting its target

- Assume

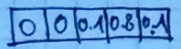
$U=2$

motion model:

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



after moving

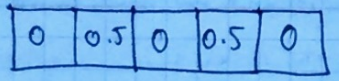


$\Rightarrow$

- The robot movement added some uncertainty to robot's position! !

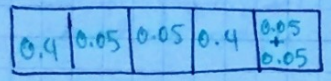
- Another example

prior



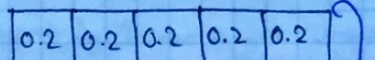
$U=2$

posterior



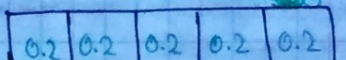
- Another example: Uniform dist.

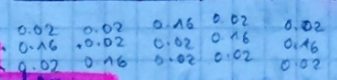
prior



$U=2$

posterior





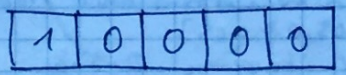
You can't get any more uncertain than the uniform dist. !

CONVOLUTION, Theorem of total probability

Limit (or Stationary) Distribution.

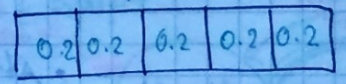
- Assume now that robot moves forever one cell at a time to right.

prior:



$U=1$ forever

after many many steps:



- As robot continues to get more and more uncertain about where it is, it will eventually reach the state of maximal uncertainty: the uniform distribution

- Every time we move we loose information !

And finally:

The distribution of absolute least information is the uniform distribution.

It has no preference what so ever!

BALANCE

Entropy:

- Entropy represents the amount of uncertainty in a system. Uniform dist. is the maximum entropy
- Entropy decreases after the measurement update step and increases after each move() step. (sense())

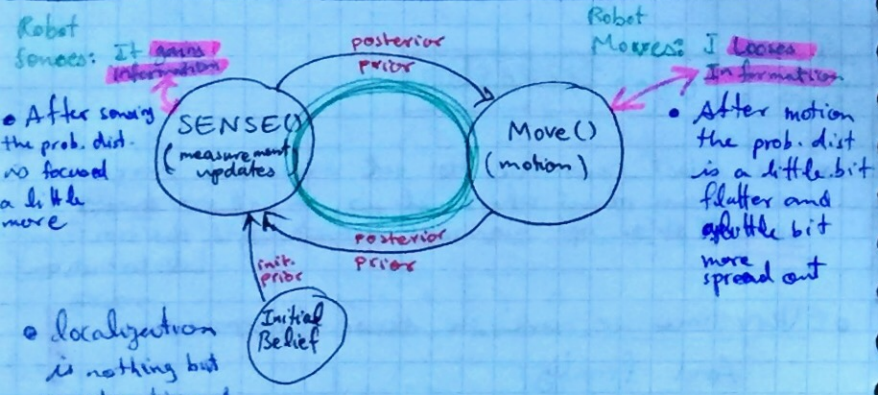
$$\text{Entropy} = \sum(-p \times \log(p))$$

[0.2, 0.2, 0.2, 0.2, 0.2]

⇒ for uniform dis (the max entropy)

$$\text{Entropy} = 5 \times (-0.2 \times \log(0.2)) = 0.699$$

- Taking measurements reduces uncertainty and hence decreases the entropy.
  Lets assume we did a measurment and probabilities became [0.05, 0.05, 0.05, 0.8, 0.05]
  now the entropy becomes 0.338



- After sensing the prob. dist. is focused a little more
- After motion the prob. dist is a little bit flatter and a little bit more spread out
- localization is nothing but an iteration of sense and move. → There is an initial belief that is tossed into the loop

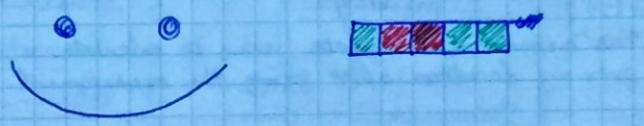
- Entropy: Expected log likelihood of the probability of each grid cell

$$\text{Entropy} = -\sum p(x_i) \log p(x_i)$$

- Entropy: a measure of information

Watch the video
Quiz: Sense & move

It is magic!!!!!



Summary:

- Localization maintains a function over all parkable places a robot might be, where each cell has an associated probability value.
- BELIEF = PROBABILITY

  SENSE = Product of Probabilities times measurements (updating) followed by normalization

  Motion = Convolution (= Addition)

  (we reversed engineered the question and asked where the robot might have come from, and added these probabilities)

Formal Definition of Localization:

Probability

$0 \leqslant P(x_i) \leqslant 1$ for x_i : grid cell ($x_i \in$ World)

$\sum_i P(x_i) = 1$

- **Measurement** : Bayes Rule

z = measurement x_i = grid cell

$$P(x_i|z) = \frac{P(z|x_i) \cdot P(x_i)}{P(z)}$$

($P(x_i)$: prior; $P(z|x_i)$: measurement probability; $P(z)$: normalizer)

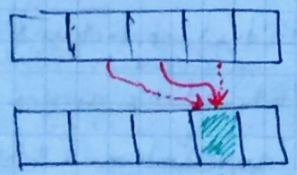
- The measurement update seeks to calculate a belief over my location after seeing the measurement.

$$P(z) = \sum_i P(z|x_i) \cdot P(x_i)$$

- **Motion**: Total Probability

$$P(X_i^t) = \sum_j P(X_j^{t-1}) \cdot P(X_i|X_j)$$

(t: time)



Convolution: weighted sum over other variable

Theorem of Total Probability: $P(A) = \sum_B P(A|B) \cdot P(B)$

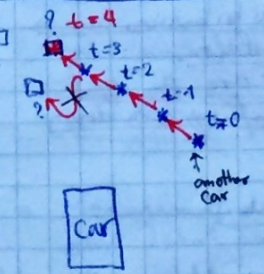
3. Working with Matrices

- **Kalman Filter**:
  An Algorithm that uses noisy senses measurements (and Bayes Rule) to produce reliable estimate of unknown quantities (like where the vehicle is likely to be in 3 seconds)

Tracking

- **Particle Filters**
  - Continous
  - Multi-modal
- **Kalman Filter**
  - Continous
  - uni-modal
- **Monte Carlo Localization**
  - Discrete
  - Multi-modal
  - Histogram Filters

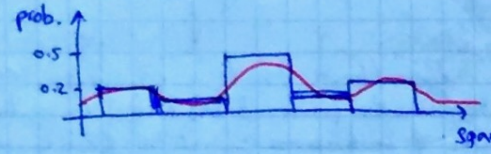
- We need to know the position, direction, and speed of other cars around us, in order to avoid colliding into them.
- We use the data from Laser-range finder for that.



Examples

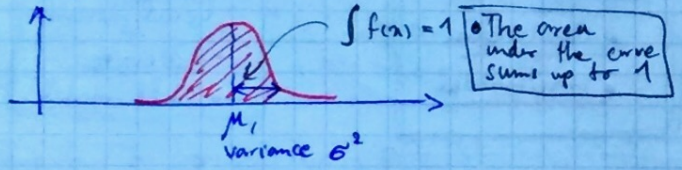
- we assume a car will go in a straight line, based on the observations t=0 to t=3.
- Similarily, a Kalman filter takes observations, and estimates future locations and velocities, even when observations are noisy and uncertain.

0.2	0.1	0.5	0.1	0.2



Histogram

- A histogram is a more approximation of a continous distribution, by dividing it into discreete partition
- The world is divided into discrete grid, and we assign to each grid cell a probability
- In Kalman filter, the distribution is given by a Gaussian. It is a continous function.

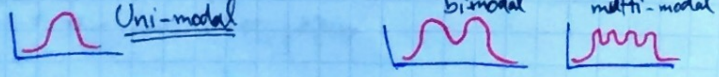


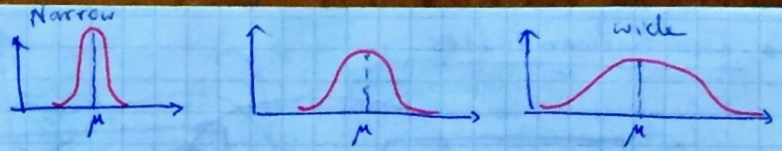
- The area under the curve sums up to 1

1-D Gaussian (μ, σ^2)

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

- Rather than estimating the entire distributions as a histogram, our task in Kalman filter is to maintain a μ and σ^2 that are the best estimate of the location of the objet we are trying to find.
- A Gaussian curve has a single peak. It is called "uni-modal"





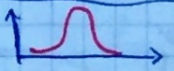
- Small σ^2
  - very certain
- medium σ^2
  - somehow certain
- large σ^2
  - uncertain

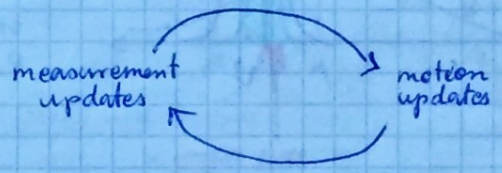
- σ^2 is a measure of certainty.

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

- normalizes the difference between x and μ.
- so the larger σ^2, the less important this diff, the less certain the dist, the more wider.

Kalman Filter





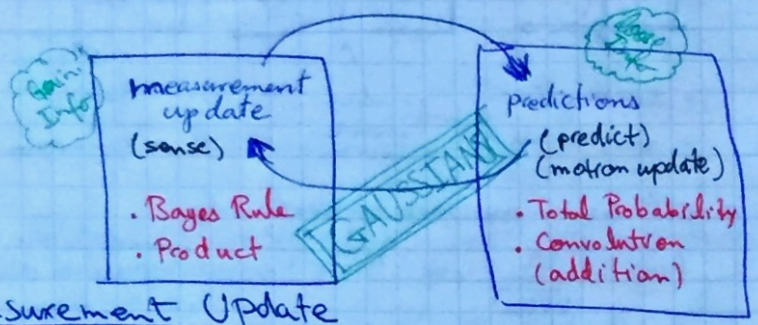
- Similar to localization, the Kalman filter iterates measurement updates and motion updates.

Quiz : In localization we had

- measurment → ☒ Product ☒ Bayes Rule
- motion → ☒ Convolution ☒ Total Probability

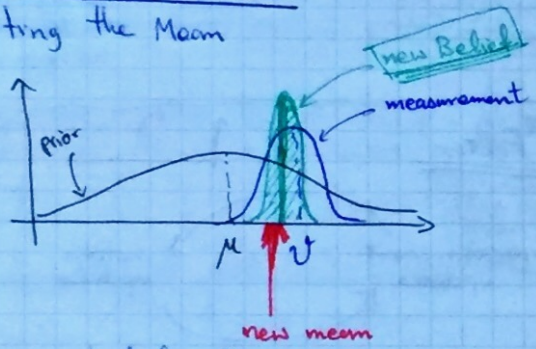
- Measurement meant updating our belief (and renormalizing our distribution)
  Motion meant keeping track of where all of our Probability "went" when we moved (which meant summing up by where we could have come from using the law of Total Probability)

- In Kalman Filter



Measurement Update

- Shifting the Mean

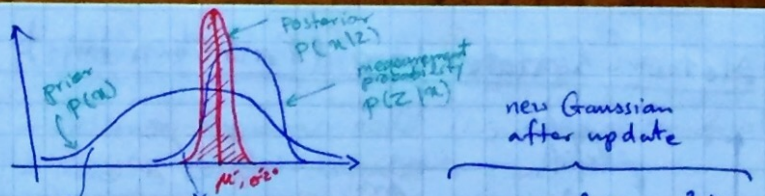


- The new belief will be somewhere between the previous belief and the new measurement.

- The new belief will be more certain than either the previous belief or the new measurement!!
  - Because the measurement can be noisy and wrong.

Take away lesson: more measurements means greater certainty.

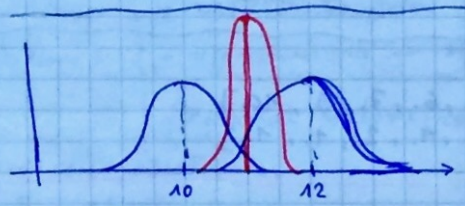
- The resulting Gaussian is more certain than the two component Gaussians.
  Because we actually gain information!!
  The two Gaussians together have higher information content than either one alone.



new Gaussian after update

$$\mu' = \frac{r^2\mu + \sigma^2\nu}{r^2 + \sigma^2}$$

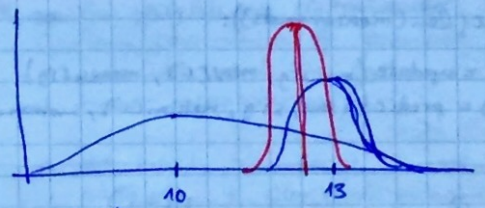
$$\sigma'^2 = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}}$$



$\mu = 10$, $\sigma^2 = 4$; $\nu = 12$, $r^2 = 4$

$\mu' = 10$

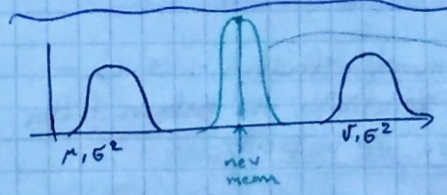
$\sigma'^2 = 2$



$\mu = 10$, $\sigma^2 = 8$; $\nu = 13$, $r^2 = 2$

$\mu' = 12.4$

$\sigma'^2 = 1.6$

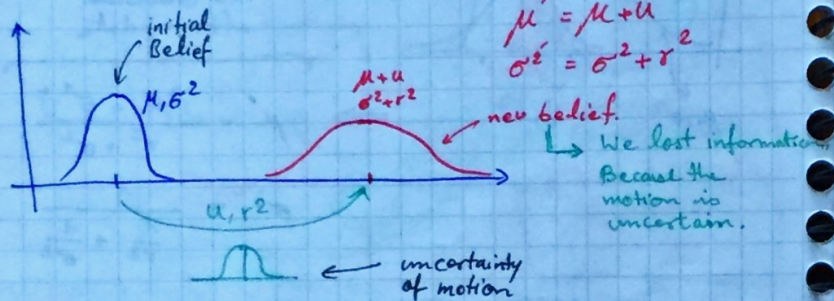


$$\sigma'^2 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{\sigma^2}} = \frac{1}{\frac{2}{\sigma^2}} = \frac{\sigma^2}{2}$$

σ'^2 does not depend on means!

- Because the Gaussians have the same width (which means the same certainty), the resulting Gaussian will be in the middle.

Motion Update Step (Prediction Step)



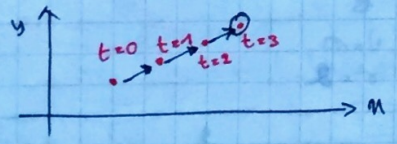
$\mu' = \mu + u$

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

↳ We lost information because the motion is uncertain.

Kalman Filter 1-D

```
measurements = [5., 6., 7., 9., 10.]
motion       = [1., 1., 2., 1., 1.]
measurement_sig = 4.
motion_sig = 2.
# intial position guess
mu = 0
sig = 1000          ~~> # very uncertain about our position guess!
for i in range(len(measurements)):
    [mu, sig] = update(mu, sig, meas[i], meas_sig)
    [mu, sig] = predict(mu, sig, motion[i], motion_sig)
```



n-D Kalman Filter

- Although we only measure location, but by considering multiple locations, the Kalman filter gives us an estimate of velocity

Kalman Filter [From Wikipedia]

Input:
- System's dynamic model (e.g. physical laws of motion)
- known control inputs to that system
- multiple sequential measurements (e.g. from sensors)

↳ Output: estimate of system state. (carrying variables)

Problems:
- noisy sensor data
- approximations in system control equations
- external factors not accounted for

Kalman Filter:

repeat recursively → estimate of Sys. state := A weighted average of sys.'s predicted state and the new measurement.

- the weights are determined from covariance of previous state estimate and the covariance of sensor measurements.
- At each time step, we use previous estimate + current measurement
- Kalman filter only requires the "last best guess" rather than entire history of sys. state
- Kalman gain: relative weight given to the measurements and current estimate.

gain ↑ ⇒ jumpy, responsive to measurement

gain ↓ ⇒ follow model predictions more closely, smooth out noise, but less responsive to new measurement.

Example: Tracking position of a truck

measurement:
- GPS → noisy, jumps within some meters
- GPS covariance: describes the certainty of measurement telling the location

movement model (Physical model of system):
- dead reckoning
- determine new position by direction and wheel rotations and ~~of~~ velocity (accel?)
- covariance: uncertain at high speed
  - drifts from real position, because of error accumulation

initial prediction

⇒ recursively
- predict ← the location using dead reckoning laws
- update ← the location estimate (prediction) using GPS measurement.

— use both covariances to weight the combination.

— The GPS measurement should pull the position estimate back towards the real position, but not disturbe it to the point of becoming rapidly jumping and noisy.

For simplicity, assume all matrices are constant
* in reality they also depend on time: $F \rightsquigarrow F_k$

F: the state transition model
H: observation model
Q: covariance of process noise
R: covariance of observation noise
B: (sometimes) control input model

Kalman Filter assumes true state at time k is evolved from previous state at $k-1$ according to:

$$x_k = F x_{k-1} + B u_k + w_k \quad \text{where } w_k \sim \mathcal{N}(0, Q)$$

process noise

At time k observation, or measurement of the true state x_k is given by:

$$z_k = H x_k + v_k \quad \text{where } v_k \sim \mathcal{N}(0, R)$$

observation noise

$\hat{x}_{n|m}$ = estimate of x at time n, given observations up to and including $m \leq n$

Predict:

$$\hat{x}_{k|k-1} = F \hat{x}_{k-1|k-1} + B u_k$$

predicted (a priori) state estimate

$$P_{k|k-1} = F P_{k-1|k-1} F^T + Q$$

predicted (a priori) error covariance

Update

$$\tilde{y}_k = z_k - H \hat{x}_{k|k-1}$$

innovation (or measurement pre-fit residual)

$$S_k = R + H P_{k|k-1} H^T$$

innovation (or pre-fit residual) covariance

$$K = P_{k|k-1} H^T S^{-1}$$

optimal Kalman gain

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K \tilde{y}_k$$

updated (a posteriori) state estimate

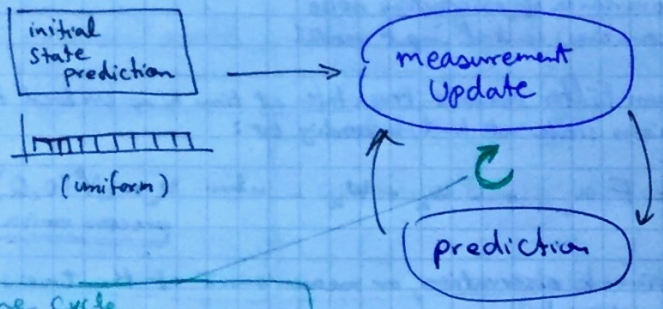
$$P_{k|k} = (I - KH) P_{k|k-1}$$

updated (a posteriori) estimate covariance

$$\tilde{y}_{k|k} = z_k - H \hat{x}_{k|k}$$

measurement post-fit residual

- The Kalman Filter combines the inaccurate sensor measurement with the inaccurate prediction of motion to get a filtered location value that is better than any estimate that comes from only sensor or only knowledge about movement.



State

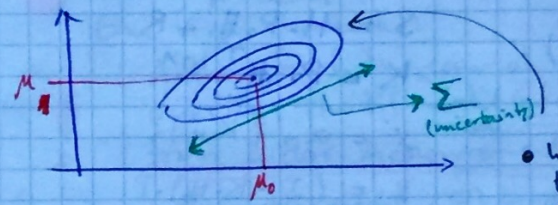
$[0, 50\ m/s)$ ⟶ initial state

- Motion model: distance = v . t
- $dx = \frac{1}{2} a\, dt^2 + V_0\, dt$
- State transformation matrix

Multivariate Gaussians

$$\mu = \begin{pmatrix} \mu_0 \\ \vdots \\ \mu_D \end{pmatrix} \quad \Sigma = \begin{pmatrix} & \\ & \end{pmatrix}$$ covariance matrix

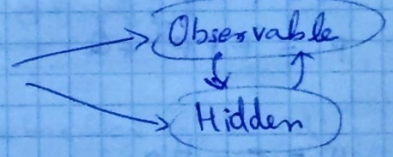
PDF: $$f_X(x_1, \ldots, x_D) = (2\pi)^{-D/2} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$



2D

- When Gaussian is tilted, this means there is corelation between uncertainty of x and y.

Kalman Filter States



Design of a KF:

$$\begin{pmatrix} x' \\ \dot{x}' \end{pmatrix} \leftarrow \underbrace{\begin{pmatrix} \text{State transformation Function} \end{pmatrix}}_{F} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$ (or state transition function) — states

$$z \leftarrow \underbrace{\begin{pmatrix} \text{measurement function} \end{pmatrix}}_{H} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$ — measurements

prediction

$x' = Fx + u$

$P' = F \cdot P F^T$

- x: estimate
- P: uncertainty covariance
- F: state transition matrix
- u: motion vector
- Q: covariance (uncertainty) of process noise
- z: measurement
- H: measurement func. (maps states to measurements)
- R: measurement noise

measurment update

Projecting system uncertainty into the measurement space

error → $y = z - H$

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

Kalman Gain → $K = P \cdot H^T \cdot S^{-1}$

$x' = x + (Ky)$

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

Simplified Equations of KF:

(sometimes) needed control input model

predict (motion update)

$x' = Fx + Bu$

$P' = FPF^T + Q$

update (measure update)

$y = z - Hx'$

$S = R + HP'H^T$

$K = P'H^T S^{-1}$

$x = x' + Ky$

$P = (I - KH)P'$

$y = z - Hx$

- Check The Jupyter Notebook on Kalman Filter. This is cool! :)
- VERY INTERESTING:

- Kalman Filters can give us insight into variables that we cannot measure directly

Ex. (From notebook):

Although lidar cannot measure velocity directly, the KF can infer the velocity from lidar measurements. (because the motion model (F) models the relationship between distance and velocity)

CPP Getting Started

- typedef vector<vector<float>> t_grid;
- function declaration vs. function definition
- #include <iostream> vs. #include "distance.h"

→ double quote looks for the files in the "main.cpp" directory first.
Brackets first looks for files where system header files are kept.

L2 - CPP Vectors (2 h 18 min)

- CPP Lists and CPP vectors → Sequence Containers

- using namespace std;
- std::vector<int> myvector (10,6)
- C++11:
  std::vector<float> myvector = {5.0, 3.0, 27, 8.2, 7.9}
- C++98, C++03, C++11, C++14, C++17

L3: Practical CPP: (59 min)

- ISO C++ Standards: C++98, C++03, C++11, C++14, C++17
- g++ -std=c++11 main.cpp
- cin >> integertwo;
- stringstream ifstream matrixfile ("matrix.txt");
- while (getline (matrixfile, line)) { — }
  - ss >> i; // next value (Also parses!)
  - ss.str(line)
  - ofstream outfile;
  - outfile.open("matrixoutput.txt");
  - outfile << i << " ";

L4: C++ OOP (2h 23m)

- include guards
- recommendation:
  "put the minimum number #include statements needed into a header file"
- include guards: prevent redeclaration of things when header files

← are included

```
#ifndef ENGINE_H
#define ENGINE_H
        .
        .
        .
#endif /* ENGINE_H */
```

- Recommendation:
  "Avoid using namespaces in header files. This can help avoid naming conflicts"

- std::vector<int>::size_type
  - is actually an unsigned int
  - is guaranteed to hold up to the maximum size of a vector.

L5: Python vs. C++ Speed

C++ is at least
- Trade-off between
  - speed of execution twice as fast
  - speed of development as Python

- Very fun way of initializing a vector from an array

```
static const arr[] = {16,22,7,69};
vector<int> vec (arr, arr + sizeof(arr)/sizeof(arr[0]));
```

:)!

(arr: pointer; arr +: pointer; sizeof(arr)/sizeof(arr[0]): length of array)

↓ user the constructor vector(iterator begin, iterator end);

Performance Programming in C++

L1: Intro to optimization (1h 40m)

- Code optimization:
  - use less memory
  - increase power efficiency
  - execute faster

`-S` only preprocess, compile, assemble steps

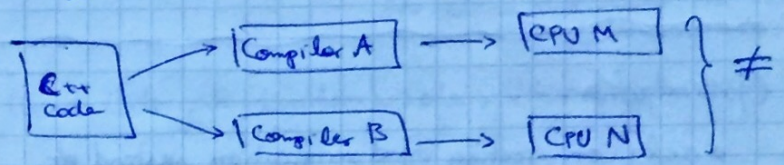
- `g++ -c main.cpp`
- `xxd main.o` ← Hex dump
  `xxd -b main.o`
- `std::bitset<8>('a')` ⇝ 0110 0001
  `std::bitset<sizeof(char)*8>(97)` ⇝ 0110 0001
-
```
float example = 97.0;
char binary[sizeof(float)];

memcpy(binary, &example, sizeof(float));
for (int i=0; i<sizeof(float); i++){
    std::cout << std::bitset<sizeof(char)*8>(binary[i]);
}
```
↓ binary representation of 97.0

- Heap vs. Stack:

`&`: address-of operator
`*`: dereference operator
`*foo`: (value pointed to by foo)

```
int * pointerVar;
pointerVar = new int;
*pointerVar = 10;
delete pointerVar;
pointerVar = NULL;
```


- Code optimization:
  - Analyzing algorithms
  - Understanding how a computer executes instructions
  - Learning the nuances of both programming language and compiler you are using.
    and Hardware



L2: Optimization Practice (2h 30 min)

- Hardware:
  - ↳ HW Limitations, e.g. CPU's slow software approximation of trigonometric functions
    - ↳ e.g. use small angle approximation if possible.
  - → Or, e.g. using 64-bit int on a 16-bit or 32-bit architecture might be inefficient.
- Compiler
  - e.g. compiler unrolls the for loop. ← to avoid checking the condition.

  [see list of possible compiler optimizations at Wikipedia]
- Algorithms
  - e.g. quicksort vs. bubble sort
  - → Check the underlying Algorithm when using STL

- fundamental type: (e.g. bool, double, ...)
- & reference variable ⇝ a variable alias
  && (since C++11 rvalue reference)
- rvalue (can't change) (temporary)
  lvalue (can change) (variable)

Optimization:

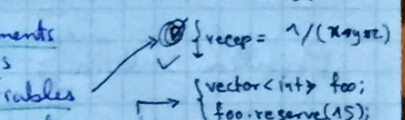
- rely on testing and verification instead of instinct!
- Performance testing, Performance analysis (Profiling)
  Algorithmic optimization
  Platform independent optimization
  SW Platform dependent optimization
  HW Platform dependent optimization

- Principle of diminishing returns
- Principle of diminishing portability

- C++ Standard Clock

```
#include <ctime>
std::clock_t start;
double duration;
start = std::clock();

function_name (var1, var2);

duration = (std::clock() - start) / (double) CLOCKS_PER_SEC;
std::cout << "duration in milliseconds" << 1000 * duration << endl;
```


- Some Optimization Techniques:
  - remove dead code
  - avoid extra if Statements
  - avoid nested for loops
  - avoid creating extra variables
  - reserve space in memory for vectors
  - passing variables by reference
  - using static → use static in functions for values that remain the same between function calls.



- Dead Code, Redundant Code, Unreachable code

- If statements:
  - put most common cases first
  - get rid of branches

- gcc -O3 flag ⇝ 150 ~ 20 ms!

  Optimization ⇝ 350 ⇝ 150 ms
  + -O3 : ⇝ 20 ms!

17 times faster 1700%

Navigating Data Structures

L1: How to Solve Problems: (2 h)

~~Compression~~

tar -cvfz ⟶ tar gz
tar -xvfz ⟶ extract.

!tar cvfz notebook.tar.gz *

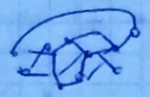
L3: The search Problem:

Problem Solving:

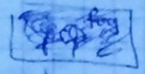
The theory and technology of building agents that can plan ahead to solve problems.

- We talk about a class of problems where the complexity of the problem comes from the idea that there are many states

  e.g. a navigation problem.
  ↳ sequence of actions



- This is in contrast to e.g. a problem of partial observability, where the actions, paths, results, ... are not fully known.



- **Formal Definition of Problem:**

  Initial State ⟶ S_0

  Actions(s) ⟶ $\{a_1, a_2, a_3, \dots\}$ (can depend on state)

  Result(s, a) ⟶ s'

  GoalTest(s) ⟶ True | False

  PathCost($s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} s_{i+2} \dots$) (mostly an additive cost → $\sum$ step costs)
  ↳ Cost value (n) where $i = 0, 1, \dots$ $s = 1, 2, \dots$

  StepCost(s, a, s') ⟶ n

→

- **State Space**: the set of all possible states
- We navigate the state space by applying actions.
- **Frontier**: the set of farthest states we have explored.
  ↳ At any given time we have { Frontier, Unexplored region, Explored region

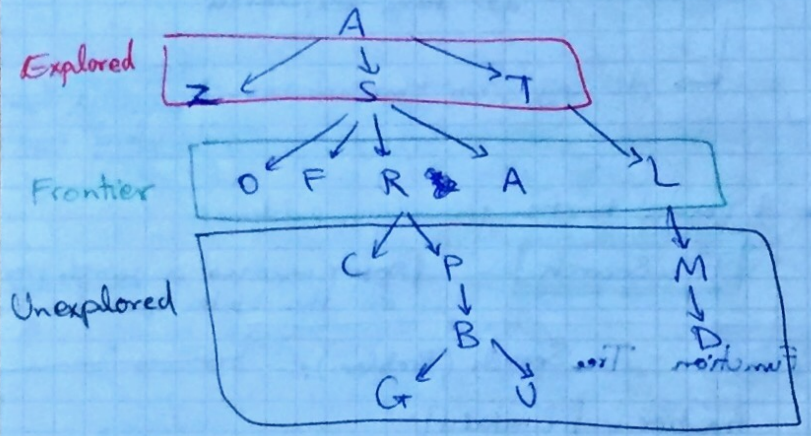
- A general function for solving problems:

  Tree Search (superimposes a search tree on the state space)

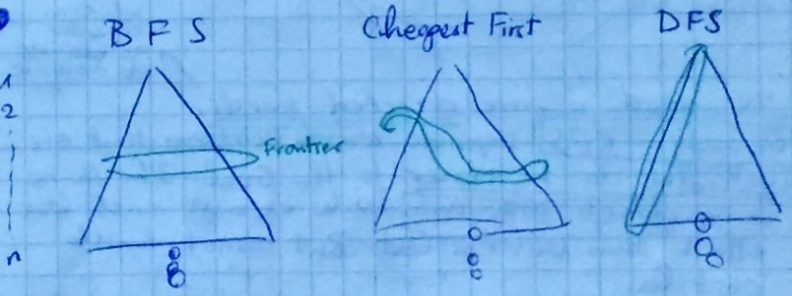
```
Function TreeSearch (Problem):
    frontier = { [initial] }
    loop:
        if frontier is empty: return FAIL
        path = remove_choice (frontier)
        s = path.end
        if s is a goal: return Path
        for a in actions(s):                      } expanding the path
            add [path + a ⟶ Result(s, a)] to frontier
```

- It is a family of algorithm:
  - they all look at frontier, →
  - pop one off
  - look if it is the goal and if yes return it
  - Otherwise expand the frontier

- What in tree Search algorithms differs is how to choose a path from frontier. (choice)
- BFS (Breadth-First Search)
  (or could be called shortest-First)



- Graph Search
  ↳ like Tree Search, but don't add states that are in Frontier or are already explored, again to the Frontier.
- BFS is guaranteed to find the path.
  But shortest path only if the number of step is considered.

- Uniform Cost Search (Dijekstra)
  (could be called "cheapest first")
- if the cost was not number of steps.
- Cheapest First (Uniform Cost Search)
  and
  BFS (for Shortest First)
  are optimal ⇝ they are guaranteed to find the shortest path
- But DFS is not optimal, it may find a longer path than the shortest one.
- 

Search Method	Optimal	Frontier Size	Size @ n=20
BFS	Yes	2^n	1,048,576
Cheapest First	Yes	2^n	1,048,576
DFS	No	n	20

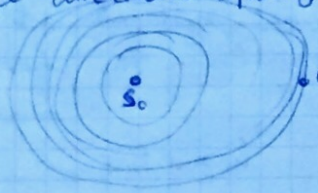
- **Completeness:**

  If there is a goal G, will the algorithm find it and terminate?

[Imagine the depth of tree is infinite]

- BFS and Cheapest-First (Uniform Cost) are compelete. DFS is not guaranteed to find the solution.

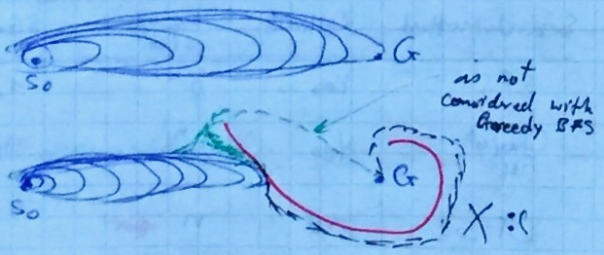
- Uniform Cost Search and BFS expand out in all directions equally



- We need a more directed search. (We can consider an estimate of distance)

Greedy BFS expands towards locations that are estimated to be closer to the Goal. If there is a direct path available expands much less effort than Uniform Cost. But cannot handle obstacles.

Greedy BFS



- We need something that combines both good features:

  - → is directed search towards G and only in many cases explores small number of nodes
  - → can handle obstacles and always gives the shortest path

- **A* Search**

it could be called: A* Best estimated total path cost first

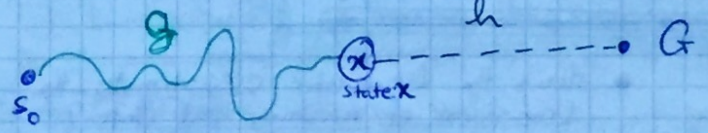
always expand the path that has a minimum value of the function f

$$f = g + h$$

$g(path)$ = path cost

$h(path) = h(s)$ = estimated distance to the Goal

(s = final state of the path)



$$f = g + h$$

- g: Cost of the path so far
- h: estimate distance to G

- minimize g ⇝ keep the path short
- minimize h ⇝ keep directed towards goal

- A* finds the lowest cost path if:

  It depends on the h function

  $h(s) \leq$ true cost

  - h should never overestimate distance to goal
  - h is optimistic ≈
  - h is admisable ≈

    → Wikipedia: Admissable Heuristic

- For an intuition on the proof, see video.
  But the idea is, since h is always smaller than actual cost, and since we pick the cheapest first path from frontier, and at the goal $h=0$, then all other pathes in the frontier must have a higher cost than the returned path p.

  i.e. C is the actual cost of P

- **Admissable heuristics**
  - does the heuristic overestimate the true cost of reaching goal state?

    → If yes, it is not admisable

Example

Fifteen Puzzle

1	2	3	4
5	6	7	8
9	11		12
13	10	14	15

h_1 = # misplaced blocks

h_2 = sum(distance of blocks)

- h_1 ⇝ Harming Distance
- h_2 ⇝ Manhatan Distance

- Both heuristics are admissable !
  → they are (less than actual or equal) number of moves needed to reach the goal!

  ≃ Both don't overestimate the actual cost.

- But $h_2 \geq h_1$ ⇒ h_2 is better and expands faster (i.e. with fewer steps) and is closer to perfect heuristic

---

- But how to automatically come up with heuristics? !

  → By relaxing the problem!

- Formal problem/solution rule & description

  A block can move A → B
  if (A ~~is adjacent to~~ B) relaxation r2
  and (~~B is Blank~~) relaxation r1

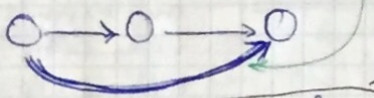
- r1 ⇝ gives us h2
- r2 ⇝ gives us h1

→

Relaxing the problem

- Take the harder problem
  - → relax ~~one of~~ one or more of the constraints
    - → you have an easier problem
      - → whose cost is less than or equal the original problem
        - → you have a heuristic that is admissable

- This is like adding new links to the state space.



that can traverse the state space easier.

- adding new operators (links) always makes the problem easier, and thus does not overestimate the cost, and so are admissable.

- When does search not work? What search can't do?

Search only works if:

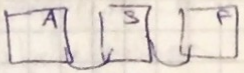
- Domain is fully Observable
- Domain is Known
- Domain is Discrete
- Domain is Deterministic
- Domain is Static

- Fully Observable: we must see what initial state we start with
- Known : we must know the set of available actions at each state
- Discreet : finite number of actions to choose from
- Deterministic : we must know the result of taking each action
- Static : nothing else can change the world except our actions.

- Note on Impl: A → S → F

Path is a linked list of Nodes.

Node: { State at the end of path F
Action (S, F)
cost 234
parent S



Frontier:

Operations: remove best item
add new one
membership test

Impl : Priority Queue

Representation: Set

Build : Hash table
Tree

Explored

add new member
check for membership

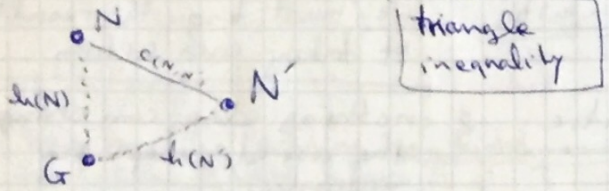
Set
Hash table
Tree

- **Consistent heuristic** $\Rightarrow$ Monotony

N : node
N' : successor

$h(N) \leq c(N, N') + h(N')$

triangle inequality



- Every consistent heuristic is also admissible. The other way is not true though.

Projects:

1. ✓ Joy Ride
2. ✓ Histogram Filter
3. ✓ Matrix class
4. ✓ Translate Python to Cpp
5. ✓ Optimize Histogram Filter (Cpp)
6. ✓ Path Planner
7. ✓ Like In (July 19th)
8. ✓ GitHub (July 21st)
9. Traffic Light Classifier (August 18th)

From "knowledge" about relations of space to "know how" to interact properly with space.

- Behavior-Based Robotics ≈ Strong Spatial Cognition
- where does these heuristics come from?

Vehicle Motion and Control

L1 Odometers, Speedometers, and Drivatives (1:30 hours)

- Shai Magzimof CEO of Phantom Auto
  Ben Shmkman, Software Engineer.

- **Odometry:** Using data from motion sensors and integrating it in order to come to a coherent belief about how far we have traveled.

- Understanding Motion: position, velocity, acceleration and their relationship.

- Derivatives, Integrals } from Calculus

- Take car's odometry data (e.g. distance traveled) and use it to infer knowledge about velocity & acceleration

- IMU: Inertial Motion unit
  - acceleration along x, y, z
  - rotation rates around x, y, z (pitch, roll, yaw)
- Vehicle heading and displacement
- Navigation Sensors
  - **Odometers**:
    Measures how far the vehicle has traveled by counting the wheel rotations.
    Useful for measuring distance traveled (displacement) but are biased because of changing tire diameters
    "trip odometer" can be reset by vehicle operator.
  - **IMU**
    measures heading, rotation rate, and linear acceleration using magnetometer, rate gyros, accelerometer

- Average Speed: $v_{avg} = \frac{\Delta a}{\Delta t}$

(1:30 h)

L2: Accelerators, rate gyros and Integrals

- Use integral of acceleration to find velocities and integral of velocities to find displacement.
- **Rate Gyros** measure **angular velocity** θ
  Integrating the angular velocity gives us heading
- yaw rate: $\dot{\theta}$

- Real Data is always flawed!
- accelerometers are often **biased**
  - → careful calibration
  - → better sensors

  i.e. → they measure some non-zero value when the acceleration is zero.

- The effect of **Error Accumulation** is drastic when integrating over longer time intervals and/or double integrating!
- In order to avoid error accumulation, we should use acceleration data only over short time intervals.